

# Rewritability in Monadic Disjunctive Datalog, MMSNP, and Expressive Description Logics

Cristina Feier, Antti Kuusisto and Carsten Lutz

Fachbereich Informatik, University of Bremen, Germany

**Abstract.** We study rewritability of monadic disjunctive Datalog programs, (the complements of) MMSNP sentences, and ontology-mediated queries (OMQs) based on expressive description logics of the  $\mathcal{ALC}$  family and on conjunctive queries. We show that rewritability into FO and into monadic Datalog (MDLog) are decidable, and that rewritability into Datalog is decidable when the original query satisfies a certain condition related to equality. We establish 2NEXPTIME-completeness for all studied problems except rewritability into MDLog for which there remains a gap between 2NEXPTIME and 3EXPTIME. We also analyze the shape of rewritings, which in the MMSNP case correspond to obstructions, and give a new construction of canonical Datalog programs that is more elementary than existing ones and also applies to formulas with free variables.

## 1 Introduction

In data access with ontologies, the premier aim is to answer queries over incomplete and heterogeneous data while taking advantage of the domain knowledge provided by an ontology [12, 19]. Since traditional database systems are often unaware of ontologies, it is common to rewrite the emerging ontology-mediated queries (OMQs) into more standard database query languages. In fact, the DL-Lite family of description logics (DLs) was designed specifically so that any OMQ  $Q = (\mathcal{T}, \Sigma, q)$  where  $\mathcal{T}$  is a DL-Lite ontology,  $\Sigma$  a data signature, and  $q$  a conjunctive query, can be rewritten into an equivalent first-order (FO) query that can then be executed using a standard SQL database system [3, 20]. In more expressive ontology languages, it is not guaranteed that for every OMQ there is an equivalent FO query. For example, this is the case for DLs of the  $\mathcal{EL}$  and Horn- $\mathcal{ALC}$  families [25, 40], and for DLs of the expressive  $\mathcal{ALC}$  family. In many members of the  $\mathcal{EL}$  and Horn- $\mathcal{ALC}$  families, however, rewritability into monadic Datalog (MDLog) is guaranteed, thus enabling the use of Datalog engines for query answering. In  $\mathcal{ALC}$  and above, not even Datalog-rewritability is generally ensured. Since ontologies emerging from practical applications tend to be structurally simple, though, there is reason to hope that (FO-, MDLog-, and Datalog-) rewritings do exist in many practically relevant cases even when the ontology is formulated in an expressive language. This has in fact been experimentally confirmed for FO-rewritability in the  $\mathcal{EL}$  family of DLs [29], and it has

led to the implementation of rewriting tools that, although incomplete, are able to compute rewritings in many practical cases [30, 39, 43].

Fundamental problems that emerge from this situation are to understand the exact limits of rewritability and to provide (complete) algorithms that decide the rewritability of a given OMQ and that compute a rewriting when it exists. These problems have been addressed in [10, 11, 29] for DLs from the  $\mathcal{EL}$  and Horn- $\mathcal{ALC}$  families. For DLs from the  $\mathcal{ALC}$  family, first results were obtained in [13] where a connection between OMQs and constraint satisfaction problems (CSPs) was established that was then used to transfer decidability results from CSPs to OMQs. In fact, rewritability is an important topic in CSP (where it would be called definability) as it constitutes a central tool for analyzing the complexity of CSPs [23, 24, 26, 32]. In particular, rewritability of (the complement of) CSPs into FO and into Datalog is NP-complete [6, 21, 32], and rewritability into MDLog is NP-hard and in EXPTIME [21]. In [13], these results were used to show that FO- and Datalog-rewritability of OMQs  $(\mathcal{T}, \Sigma, q)$  where  $\mathcal{T}$  is formulated in  $\mathcal{ALC}$  or a moderate extension thereof and  $q$  is an atomic query (AQ) of the form  $A(x)$  is decidable and, in fact, NEXPTIME-complete. For MDLog-rewritability, one can show NEXPTIME-hardness and containment in 2EXPTIME.

The aim of this paper is to study the above questions for OMQs where the ontology is formulated in an expressive DL from the  $\mathcal{ALC}$  family and where the actual query is a conjunctive query (CQ) or a union of conjunctive queries (UCQ). As observed in [13], transitioning in OMQs from AQs to UCQs corresponds to the transition from CSP to its logical generalization MMSNP introduced by Feder and Vardi [26] and studied, for example, in [14, 34–36]. More precisely, while the OMQ language  $(\mathcal{ALC}, \text{AQ})$  that consists of all OMQs  $(\mathcal{T}, \Sigma, q)$  where  $\mathcal{T}$  is formulated in  $\mathcal{ALC}$  and  $q$  is an AQ has the same expressive power as the complement of CSP (with multiple templates and a single constant), the OMQ language  $(\mathcal{ALC}, \text{UCQ})$  has the same expressive power as the complement of MMSNP (with free variables)—which in turn is a notational variant of monadic disjunctive Datalog (MDDLog). It should be noted, however, that while all these formalisms are equivalent in expressive power, they differ significantly in succinctness [13]; in particular, the best known translation of OMQs into MMSNP/MDDLog involves a double exponential blowup. In contrast to the CSP case, FO-, MDLog-, and Datalog-rewritability of (the complement of) MMSNP sentences was not known to be decidable. In this paper, we establish decidability of FO- and MDLog-rewritability in  $(\mathcal{ALC}, \text{UCQ})$  and related OMQ languages, in MDDLog, and the complement of MMSNP. We show that FO-rewritability is 2NEXPTIME-complete in all three cases, and that MDLog-rewritability is in 3EXPTIME; a 2NEXPTIME lower bound was established in [17]. Let us discuss our results on FO-rewritability from three different perspectives. From the OMQ perspective, the transition from AQs to UCQs results in an increase of complexity from NEXPTIME to 2NEXPTIME. From the monadic Datalog perspective, adding disjunction (transitioning from monadic Datalog to MDDLog) results in a moderate increase of complexity from 2EXPTIME [8] to 2NEXPTIME. And

from the CSP perspective, the transition from CSPs to MMSNP results in a rather dramatic complexity jump from NP to 2NEXPTIME.

For Datalog-rewritability, we obtain only partial results. In particular, we show that Datalog-rewritability is decidable and 2NEXPTIME-complete for MDDLog programs that, in a certain technical sense made precise in the paper, *have equality*. For the general case, we only obtain a potentially incomplete procedure. It is well possible that the procedure is in fact complete, but proving this remains an open issue for now. These results also apply to analogously defined classes of MMSNP sentences and OMQs that have equality.

While we mainly focus on deciding whether a rewriting exists rather than actually computing it, we also analyze the shape that rewritings can take. Since the shape turns out to be rather restricted, this is important information for algorithms (complete or incomplete) that seek to compute rewritings. In the CSP/MMSNP world, this corresponds to analyzing obstruction sets for MMSNP, in the style of CSP obstructions [4, 18, 37] and not to be confused with colored forbidden patterns sometimes used to characterize MMSNP [36]. More precisely, we show that an OMQ  $(\mathcal{T}, \Sigma, q)$  is FO-rewritable if and only if it is rewritable into a UCQ in which each CQ has treewidth  $(1, \max\{2, n_q\})$ ,  $n_q$  the size of  $q$ ;<sup>1</sup> equivalently, the complement of an MMSNP sentence  $\varphi$  is FO-definable if and only if it admits a finite set of finite obstructions of treewidth  $(1, k)$  where  $k$  is the diameter of  $\varphi$  (the maximum size of a negated conjunction in its body, in Feder and Vardi's terminology). We also show that an OMQ  $(\mathcal{T}, \Sigma, q)$  is MDLog-rewritable if and only if it is rewritable into an MDLog program of diameter  $(1, \max\{2, n_q\})$ ; equivalently, the complement of an MMSNP sentence  $\varphi$  is MDLog-definable if and only if it admits a (potentially infinite) set of finite obstructions of treewidth  $(1, k)$  where  $k$  is the diameter of  $\varphi$ . For the Datalog case, we give a new and direct construction of canonical Datalog-rewritings. It has been observed in [26] that for every CSP and all  $\ell, k$ , it is possible to construct a canonical Datalog program  $\Pi$  of width  $\ell$  and diameter  $k$  in the sense that if any such program is a rewriting of the CSP, then so is  $\Pi$ ; moreover, even when there is no  $(\ell, k)$ -Datalog rewriting, then  $\Pi$  is the best possible approximation of such a rewriting. The existence of canonical Datalog-rewritings for (the complement of) MMSNP sentences was already known from [15]. However, the construction given there is quite intricate, proceeding via an infinite template that is obtained by applying an intricate construction due to Cherlin, Shelah, and Shi [22], which makes them rather hard to analyze. In contrast, our construction is elementary and essentially parallels the CSP case; it also applies to MMSNP formulas with free variables, where the canonical program takes a rather special form that involves *parameters*, similar in spirit to the parameters to least fixed-point operators in FO(LFP) [7].

Our main technical tool is the translation of an MMSNP sentence into a CSP exhibited by Feder and Vardi [26]; actually, the target of the translation is

---

<sup>1</sup> What we mean here is that  $q$  has a tree decomposition in which every bag has at most  $\max\{2, n_q\}$  elements and in which neighboring bag overlap in at most one element.

a generalized CSP, meaning that there are multiple templates. The translation is not equivalence preserving and involves a double exponential blowup, but was designed so as to preserve complexity up to polynomial time reductions. Here, we are not so much interested in the complexity aspect, but rather in the semantic relationship between the original MMSNP sentence and the constructed CSP. It turns out that the translation does not quite preserve rewritability. In particular, when the original MMSNP sentence has a rewriting, then the natural way of constructing from it a rewriting for the CSP is sound only on instances of high girth. However, FO- and MDLog-rewritings that are sound on high girth (and unconditionally complete) can be converted into rewritings that are unconditionally sound (and complete). The same is true for Datalog-rewritings when the MMSNP sentence has equality, but it remains open whether it is true in the general case.

The structure of this paper is as follows. In Section 2, we introduce the essentials of disjunctive Datalog and its relevant fragments as well as CSP and MMSNP; in fact, we shall always work with Boolean MDDLog rather than with the complement of MMSNP. In Section 3, we summarize the main properties of Feder and Vardi’s translation of MMSNP into CSP. We use this in Section 4 to show that FO- and MDLog-rewritability of Boolean MDDLog programs and of the complement of MMSNP sentences is decidable, also establishing the announced complexity results. In Section 5, we analyze the shape of FO- and MDLog-rewritings and of obstructions for MMSNP sentences. We also establish an MMSNP analogue of an essential combinatorial lemma for CSPs that allows to replace a structure by a structure of high girth while preserving certain homomorphisms (in the CSP case) respectively the truth of certain MMSNP sentences (in the MMSNP case). In Section 6, we study Datalog-rewritability of MDDLog programs that have equality and construct canonical Datalog programs. Section 7 is concerned with lifting our results from the Boolean case to the general case, concerning the complexity of deciding rewritability, the shape of rewritings, and the construction of canonical Datalog programs. In this section, Datalog programs with parameters play a central role. In Section 8, we introduce OMQs and further lift our results to this setting. We conclude in Section 9.

## 2 Preliminaries

A *schema* is a finite collection  $\mathbf{S} = (S_1, \dots, S_k)$  of relation symbols with associated arity. An  *$\mathbf{S}$ -fact* is an expression of the form  $S(a_1, \dots, a_n)$  where  $S \in \mathbf{S}$  is an  $n$ -ary relation symbol, and  $a_1, \dots, a_n$  are elements of some fixed, countably infinite set  $\text{const}$  of *constants*. For an  $n$ -ary relation symbol  $S$ ,  $\text{pos}(S)$  is  $\{1, \dots, n\}$ . An  *$\mathbf{S}$ -instance*  $I$  is a finite set of  $\mathbf{S}$ -facts. The *active domain*  $\text{dom}(I)$  of  $I$  is the set of all constants that occur in a fact in  $I$ . For an instance  $I$  and a schema  $\mathbf{S}$ , we write  $I|_{\mathbf{S}}$  to denote the restriction of  $I$  to the relations in  $\mathbf{S}$ .

A *tree decomposition* of an instance  $I$  is a pair  $(T, (B_v)_{v \in V})$ , where  $T = (V, E)$  is an undirected tree and  $(B_v)_{v \in V}$  is a family of subsets of  $\text{dom}(I)$  such that the following conditions are satisfied:

1. for all  $a \in \text{dom}(I)$ ,  $\{v \in V \mid a \in B_v\}$  is nonempty and connected in  $T$ ;
2. for every fact  $R(a_1, \dots, a_r)$  in  $I$ , there is a  $v \in V$  such that  $a_1, \dots, a_r \in B_v$ .

Unlike in the traditional setup [27], we are interested in two parameters of tree decompositions instead of only one. We call  $(T, (B_v)_{v \in V})$  an  $(\ell, k)$ -tree decomposition if for all  $v, v' \in V$ ,  $|B_v \cap B_{v'}| \leq \ell$  and  $|B_v| \leq k$ . An instance  $I$  has treewidth  $(\ell, k)$  if it admits an  $(\ell, k)$ -tree decomposition.

We now define the notion of girth. A finite structure  $I$  has a cycle of length  $n$  if it contains distinct facts  $R_0(\mathbf{a}_0), \dots, R_{n-1}(\mathbf{a}_{n-1})$ ,  $\mathbf{a}_i = a_{i,1} \dots a_{i,m_i}$ , and there are positions  $p_i, p'_i \in \text{pos}(R_i)$ ,  $0 \leq i < n$  such that:

- $p_i \neq p'_i$  for  $1 \leq i \leq n$ ;
- $a_{i,p'_i} = a_{i \oplus 1, p_i \oplus 1}$  for  $0 \leq i < n$ , where  $\oplus$  denotes addition modulo  $n$ .

The girth of  $I$  is the length of the shortest cycle in it and  $\infty$  if  $I$  has no cycle (in which case we say that  $I$  is a tree).

A constraint satisfaction problem (CSP) is defined by an instance  $T$  over a schema  $\mathbf{S}_E$ , called template. The problem associated with  $T$ , denoted  $\text{CSP}(T)$ , is to decide whether an input instance  $I$  over  $\mathbf{S}_E$  admits a homomorphism to  $T$ , denoted  $I \rightarrow T$ . We use  $\text{coCSP}(T)$  to denote the complement problem, that is, deciding whether  $I \not\rightarrow T$ . A generalized CSP is defined by a set of templates  $S$  over the same schema  $\mathbf{S}_E$  and asks for a homomorphism from the input  $I$  to at least one templates  $T \in S$ , denoted  $I \rightarrow S$ .

An MMSNP sentence  $\theta$  over schema  $\mathbf{S}_E$  has the form  $\exists X_1 \dots \exists X_n \forall x_1 \dots \forall x_m \varphi$  with  $X_1, \dots, X_n$  monadic second-order variables,  $x_1, \dots, x_m$  first-order variables, and  $\varphi$  a conjunction of quantifier-free formulas of the form

$$\alpha_1 \wedge \dots \wedge \alpha_n \rightarrow \beta_1 \vee \dots \vee \beta_m \text{ with } n, m \geq 0,$$

where each  $\alpha_i$  takes the form  $X_i(x_j)$  or  $R(\mathbf{x})$  with  $R \in \mathbf{S}_E$ , and each  $\beta_i$  takes the form  $X_i(x_j)$ . The diameter of  $\theta$  is the maximum number of variables in some implication in  $\varphi$ . This presentation is syntactically different from, but semantically equivalent to the original definition from [26], which does not use the implication symbol and instead restricts the allowed polarities of atoms. Both forms can be interconverted in polynomial time, see [13]. More information on MMSNP can be found, e.g., in [14, 15].

A conjunctive query (CQ) takes the form  $\exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$  where  $\varphi$  is a conjunction of relational atoms and  $\mathbf{x}, \mathbf{y}$  denote tuples of variables; the equality relation may be used. Whenever convenient, we will confuse a CQ  $\exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$  with the set of atoms in  $\varphi$ . A union of conjunctive queries (UCQ) is a disjunction of CQs with the same free variables. A disjunctive Datalog rule  $\rho$  has the form

$$S_1(\mathbf{x}_1) \vee \dots \vee S_m(\mathbf{x}_m) \leftarrow R_1(\mathbf{y}_1) \wedge \dots \wedge R_n(\mathbf{y}_n)$$

where  $n > 0$  and  $m \geq 0$ . We refer to  $S_1(\mathbf{x}_1) \vee \dots \vee S_m(\mathbf{x}_m)$  as the head of  $\rho$ , and to  $R_1(\mathbf{y}_1) \wedge \dots \wedge R_n(\mathbf{y}_n)$  as the body. Every variable that occurs in the head of a rule  $\rho$  is required to also occur in the body of  $\rho$ .

A *disjunctive Datalog (DDLog) program*  $\Pi$  is a finite set of disjunctive Datalog rules with a selected *goal relation*  $\text{goal}$  that does not occur in rule bodies and appears only in non-disjunctive *goal rules*  $\text{goal}(\mathbf{x}) \leftarrow R_1(\mathbf{x}_1) \wedge \dots \wedge R_n(\mathbf{x}_n)$ . The *arity* of  $\Pi$  is the arity of the  $\text{goal}$  relation; we say that  $\Pi$  is *Boolean* if it has arity zero. Relation symbols that occur in the head of at least one rule of  $\Pi$  are *intensional (IDB) relations*, and all remaining relation symbols in  $\Pi$  are *extensional (EDB) relations*. Note that, by definition,  $\text{goal}$  is an IDB relation.

We will sometimes use body atoms of the form  $\text{true}(x)$  that are vacuously true for all elements of the active domain. This is just syntactic sugar since any rule with body atom  $\text{true}(x)$  can equivalently be replaced by a set of rules obtained by replacing  $\text{true}(x)$  in all possible ways with an atom  $R(x_1, \dots, x_n)$  where  $R$  is an EDB relation and where  $x_i = x$  for some  $i$  and all other  $x_i$  are fresh variables.

A DDLog program is called *monadic* or an *MDDLog program* if all its IDB relations with the possible exception of  $\text{goal}$  have arity at most one. The *size* of a DDLog program  $\Pi$  is the number of symbols needed to write it (where relation symbols and variables names count one), its *width* is the maximum arity of non- $\text{goal}$  IDB relations used in it, and its *diameter* is the maximum number of variables that occur in a rule in  $\Pi$ .

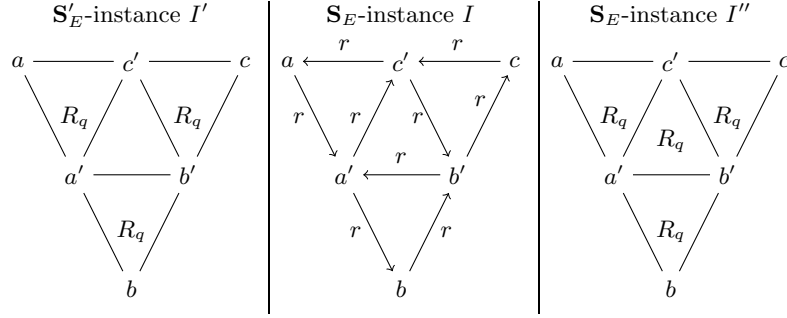
A *Datalog rule* is a disjunctive Datalog rule in which the rule head contains exactly one disjunct. Datalog (DLog) programs and monadic Datalog (MDLog) programs are then defined in the expected way. We call a Datalog program an  $(\ell, k)$ -*Datalog program* if its width is  $\ell$  and its diameter is  $k$ .

For  $\Pi$  an  $n$ -ary MDDLog program over schema  $\mathbf{S}_E$ , an  $\mathbf{S}_E$ -instance  $I$ , and  $a_1, \dots, a_n \in \text{dom}(I)$ , we write  $I \models \Pi(a_1, \dots, a_n)$  if  $\Pi \cup I \models \text{goal}(a_1, \dots, a_n)$  where the variables in all rules of  $\Pi$  are universally quantified and thus  $\Pi$  is a set of first-order (FO) sentences. A query  $q$  over  $\mathbf{S}_E$  of arity  $n$  is

- *sound for  $\Pi$*  if for all  $\mathbf{S}_E$ -instances  $I$  and  $\mathbf{a} \in \text{dom}(I)$ ,  $I \models q(\mathbf{a})$  implies  $I \models \Pi(\mathbf{a})$ ;
- *complete for  $\Pi$*  if for all  $\mathbf{S}_E$ -instances  $I$  and  $\mathbf{a} \in \text{dom}(I)$ ,  $I \models \Pi(\mathbf{a})$  implies  $I \models q(\mathbf{a})$ ;
- a *rewriting of  $\Pi$*  if it is sound for  $\Pi$  and complete for  $\Pi$ .

To additionally specify the syntactic shape of  $q$ , we speak of a UCQ-rewriting, an MDLog-rewriting, and so on. An *FO-rewriting* takes the form of an FO-query that uses only relations from the relevant EDB schema and possibly equality, but neither constants nor function symbols. We say that  $\Pi$  is *FO-rewritable* if there is an FO-rewriting of  $\Pi$ , and likewise for *UCQ-rewritability* and *MDLog-rewritability*.

It was shown in [13] that the complement of an MMSNP sentence can be translated into an equivalent Boolean MDDLog program in polynomial time and vice versa; moreover, the transformations preserve diameter and all other parameters relevant for this paper. From now on, we will thus not explicitly distinguish between Boolean MDDLog and (the complement of) MMSNP.



**Fig. 1.** Translating an  $\mathbf{S}'_E$ -instance into an  $\mathbf{S}_E$ -instance and vice versa

### 3 MDDLog, Simple MDDLog and CSPs

Feder and Vardi show how to translate an MMSNP sentence into a generalized CSP that has the same complexity up to polynomial time reductions [26]. The resulting CSP has a different schema than the original MMSNP sentence and is thus not equivalent to it. We are going to make use of this translation to reduce rewritability problems for MDDLog to the corresponding problems for CSPs. Consequently, our main interest is in the precise semantic relationship between the MMSNP sentence and the constructed CSP, rather than in their complexity. In this section, we sum up the properties of the results obtained in [26] that are relevant for us. These properties are all we need in later sections, that is, we do not need to go further into the details of the translation. For the reader's convenience and information, we still give full details in Appendix A; these are based on the presentation given in [17], which is more detailed than the original presentation in [26].

Let  $\mathbf{S}_E$  be a schema. A schema  $\mathbf{S}'_E$  is a *k-aggregation schema* for  $\mathbf{S}_E$  if its relations have the form  $R_{q(\mathbf{x})}$  where  $q(\mathbf{x})$  is a CQ over  $\mathbf{S}_E$  and the arity of  $R_{q(\mathbf{x})}$  is identical to the number of free variables of  $q(\mathbf{x})$ , which is at most  $k$ . The generalized CSP to be constructed later makes use of a schema of this form. What is important at this point is that there are natural translations of instances between the two schemas. To make this precise, let  $I$  be an  $\mathbf{S}_E$ -instance. The *corresponding  $\mathbf{S}'_E$ -instance  $I'$*  consists of all facts  $R_{q(\mathbf{x})}(\mathbf{a})$  such that  $I \models q(\mathbf{a})$ . Conversely, let  $I'$  be an  $\mathbf{S}'_E$ -instance. The *corresponding  $\mathbf{S}_E$ -instance  $I$*  consists of all facts  $S(\mathbf{b})$  such that  $R_{q(\mathbf{x})}(\mathbf{a}) \in I'$  and  $S(\mathbf{b})$  is a conjunct of  $q(\mathbf{a})$ .

*Example 1.* Assume that  $\mathbf{S}_E$  consists of the binary relation  $r$ . Let

$$q(\mathbf{x}) = r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1)$$

where  $\mathbf{x} = (x_1, x_2, x_3)$  and let  $\mathbf{S}'_E$  consist of  $R_{q(\mathbf{x})}$ . Take the  $\mathbf{S}'_E$ -instance  $I'$  defined by

$$R_q(a, a', c'), R_q(b, b', a'), R_q(c, c', b').$$

The corresponding  $\mathbf{S}_E$ -instance  $I$  is

$$r(a, a'), r(a', c'), r(c', a), r(b, b'), r(b', a'), r(a', b), r(c, c'), r(c', b), r(b', c).$$

Note that, when we take the  $\mathbf{S}'_E$ -instance  $I''$  corresponding to  $I$ , we do *not* obtain  $I'$ , but rather a strict superset that contains additional facts such as  $R_q(c', b', a')$ . The instances  $I$ ,  $I'$ , and (a subset of)  $I''$  are depicted in Figure 1.

The translation in [26] consists of two steps. We describe them here using Boolean MDDLog instead of (the complement of) MMSNP. The first step is to transform the given Boolean MDDLog program  $\Pi$  into a Boolean MDDLog program  $\Pi_S$  over a suitable aggregation schema  $\mathbf{S}'_E$  such that  $\Pi_S$  is of a restricted syntactic form. In the second step, one transforms  $\Pi_S$  into a generalized CSP whose complement is equivalent to  $\Pi_S$ .

We start with summing up the important aspects of the first step. A Boolean MDDLog program  $\Pi_S$  is *simple* if it satisfies the following conditions:

1. every rule in  $\Pi_S$  contains at most one EDB atom and this atom contains all variables of the rule body, each variable exactly once;
2. rules without an EDB atom contain at most a single variable.

Now, the first step achieves the following.

**Theorem 1 ([26]).** *Given a Boolean MDDLog program  $\Pi$  over EDB schema  $\mathbf{S}_E$  of diameter  $k$ , one can construct a simple Boolean MDDLog program  $\Pi_S$  over a  $k$ -aggregation schema  $\mathbf{S}'_E$  for  $\mathbf{S}_E$  such that*

1. *If  $I$  is an  $\mathbf{S}_E$ -instance and  $I'$  the corresponding  $\mathbf{S}'_E$ -instance, then  $I \models \Pi$  iff  $I' \models \Pi_S$ ;*
2. *If  $I'$  is an  $\mathbf{S}'_E$ -instance and  $I$  the corresponding  $\mathbf{S}_E$ -instance, then*
  - (a)  *$I' \models \Pi_S$  implies  $I \models \Pi$ ;*
  - (b)  *$I \models \Pi$  implies  $I' \models \Pi_S$  if the girth of  $I'$  exceeds  $k$ .*

*If  $\Pi$  is of size  $n$ , then the size of  $\Pi^s$  and the cardinality of  $\mathbf{S}'_E$  are bounded by  $2^{p(k \cdot \log n)}$ ,  $p$  a polynomial, and the arity of relations in  $\mathbf{S}'_E$  is bounded by  $k$ . The construction takes time polynomial in the size of  $\Pi_S$ .*

The translation underlying Theorem 1 consists of three steps itself: first saturate  $\Pi$  by adding all rules that can be obtained from a rule in  $\Pi$  by identifying variables; then rewrite  $\Pi$  in an equivalence-preserving way so that all rule bodies are biconnected, introducing fresh unary and nullary IDBs as needed. And finally replace the conjunction  $q(\mathbf{x})$  of all EDB atoms in each rule body with a single EDB atom  $R_{q(\mathbf{x})}(\mathbf{x})$ , additionally taking care of interactions between the new EDB relations that arise e.g. when we have two relations  $R_{q(\mathbf{x})}$  and  $R_{p(\mathbf{x})}$  such that  $q(\mathbf{x})$  is contained in  $p(\mathbf{x})$  (in the sense of query containment).

The following theorem summarizes the second step of the translation of Boolean MDDLog into generalized coCSP.

**Theorem 2 ([26]).** *Let  $\Pi$  be a simple Boolean MDDLog program over EDB schema  $\mathbf{S}_E$  and with IDB schema  $\mathbf{S}_I$ ,  $m$  the maximum arity of relations in  $\mathbf{S}_E$ . Then there exists a set of templates  $S_\Pi$  over  $\mathbf{S}_E$  such that*



1.  $\Pi$  is equivalent to  $\text{coCSP}(S_\Pi)$ ;
2.  $|S_\Pi| \leq 2^{|\mathbf{S}_I|}$  and  $|T| \leq |\mathbf{S}_E| \cdot 2^{m|\mathbf{S}_I|}$  for each  $T \in S_\Pi$ ;

The construction takes time polynomial in  $\sum_{T \in S_\Pi} |T|$ .

We again sketch the idea underlying the proof of the theorem. The desired set of templates  $S_\Pi$  contains one template for every 0-type, that is, for every set of nullary IDB relations in  $\Pi$  that does not contain  $\text{goal}()$  and that satisfies all rules in  $\Pi$  which use only nullary IDBs. Each template contains one constant  $c_M$  for every 1-type  $M$ , that is, for every set  $M$  of unary IDBs that agrees on nullary IDBs with the 0-type for which the template was constructed and that satisfy all rules in  $\Pi$  which use only IDBs that are at most unary. One then interprets all EDB relations in a maximal way so that all rules in  $\Pi$  are satisfied. The fact that  $\Pi$  is simple implies that there are no choices, that is, there is only one maximal interpretation of each EDB relation and the interpretations of different such relations do not interact. Details are given in the appendix.

## 4 FO- and MDLog-Rewritability of Boolean MDDLog Programs

We exploit the translations described in the previous section and the known results that FO-rewritability of CSPs and MDLog-rewritability of coCSPs are decidable to obtain analogous results for Boolean MDDLog, and thus also for MMSNP. In the case of FO, we obtain tight 2NEXPTIME complexity bounds. For MDLog, the exact complexity remains open (as in the CSP case), between 2NEXPTIME and 3EXPTIME.

We start with observing that FO-rewritability and MDLog-rewritability are more closely related than one might think at first glance. In fact, every MDLog-rewriting can be viewed as an infinitary UCQ-rewriting and, by Rossman's homomorphism preservation theorem [41], FO-rewritability of a Boolean MDDLog program coincides with (finitary) UCQ-rewritability. The latter is true also in the non-Boolean case.

**Proposition 1.** *Let  $\Pi$  be an MDDLog program. Then  $\Pi$  is FO-rewritable iff  $\Pi$  is UCQ-rewritable.*

**Proof.** It is well known and easy to show that truth of disjunctive Datalog programs is preserved under homomorphisms. Thus, the proposition immediately follows from Rossman's theorem in the Boolean case. For the non-Boolean case, we observe that Rossman establishes his result also in the presence of constants. Let  $\Pi$  be an MDDLog program and  $\varphi(\mathbf{x})$  a rewriting of  $\Pi$ . We can apply Rossman's result to  $\varphi(\mathbf{a})$ , where  $\mathbf{a}$  is a vector of constants of the same length as  $\mathbf{x}$ , obtaining a UCQ  $q(\mathbf{a})$  equivalent to  $\varphi(\mathbf{a})$ . Let  $q(\mathbf{x})$  be obtained from  $q(\mathbf{a})$  by replacing the constants in  $\mathbf{a}$  with the variables from  $\mathbf{x}$ . It can be verified that  $q(\mathbf{x})$  is a rewriting of  $\Pi$ .  $\square$

For utilizing the translation of Boolean MDDLLog programs to generalized CSPs in the intended way, the interesting aspect is to deal with the translation of a Boolean MDDLLog program  $\Pi$  into a simple program  $\Pi_S$  stated in Theorem 1, since it is not equivalence preserving. The following proposition relates rewritings of  $\Pi$  to rewritings of  $\Pi_S$ .

**Lemma 1.** *Let  $\Pi$  be a Boolean MDDLLog program of diameter  $k$ ,  $\Pi_S$  as in Theorem 1, and  $\mathcal{Q} \in \{UCQ, MDLog, DLog\}$ . Then*

1. *every  $\mathcal{Q}$ -rewriting of  $\Pi_S$  can effectively be converted into a  $\mathcal{Q}$ -rewriting of  $\Pi$ ;*
2. *every  $\mathcal{Q}$ -rewriting of  $\Pi$  can effectively be converted into a  $\mathcal{Q}$ -rewriting of  $\Pi_S$  that is (i) sound on instances of girth exceeding  $k$  and (ii) complete.*

**Proof.** Let  $\mathbf{S}_E$  and  $\mathbf{S}'_E$  be the EDB schema of  $\Pi$  and of  $\Pi_S$ , respectively. We start with the case  $\mathcal{Q} = UCQ$ .

For Point 1, let  $q_{\Pi_S}$  be a UCQ-rewriting of  $\Pi_S$ . Let  $q_{\Pi}$  be the UCQ obtained from  $q_{\Pi_S}$  by replacing every atom  $R_{q(\mathbf{x})}(\mathbf{y})$  with  $q[\mathbf{y}/\mathbf{x}]$ , that is, with the result of replacing the variables  $\mathbf{x}$  in  $q(\mathbf{x})$  with the variables  $\mathbf{y}$  (which may lead to identifications). We show that  $q_{\Pi}$  is as required. Let  $I$  be an  $\mathbf{S}_E$ -instance and  $I'$  the corresponding  $\mathbf{S}'_E$ -instance. Then we have  $I \models \Pi$  iff  $I' \models \Pi_S$  (by Point 1 of Theorem 1) iff  $I' \models q_{\Pi_S}$  (by choice of  $q_{\Pi_S}$ ) iff  $I \models q_{\Pi}$  (by construction of  $I'$  and of  $q_{\Pi}$ ). Let us expand on the latter.

First assume that  $I' \models q_{\Pi_S}$ . Then there is a CQ  $q$  in  $q_{\Pi_S}$  and a homomorphism  $h$  from  $q$  to  $I'$ . By construction,  $q_{\Pi}$  contains a CQ  $q'$  that is obtained from  $q$  by replacing every atom  $R_{q(\mathbf{x})}(\mathbf{y}) \in q$  with  $q[\mathbf{y}/\mathbf{x}]$ . Clearly, for every atom  $R_{q(\mathbf{x})}(\mathbf{y}) \in q$ , we must have  $R_{q(\mathbf{x})}(h(\mathbf{y})) \in I'$ . The construction of  $I'$  yields  $q(h(\mathbf{y})) \subseteq I$ . Consequently,  $h$  is also a homomorphism from  $q'$  to  $I$ . Conversely, assume that there is a CQ  $q'$  in  $q_{\Pi}$  and a homomorphism  $h$  from  $q'$  to  $I$ . Then there is a CQ  $q$  in  $q_{\Pi_S}$  from which  $q'$  was obtained by the described replacement operation. For every atom  $R_{q(\mathbf{x})}(\mathbf{y}) \in q$ , we must have  $q(h(\mathbf{y})) \subseteq I$ . We obtain  $R_{q(\mathbf{x})}(h(\mathbf{y})) \in q$  and thus  $h$  is a homomorphism from  $q$  to  $I'$ .

For Point 2, let  $q_{\Pi}$  be a UCQ-rewriting of  $\Pi$ . The UCQ  $q_{\Pi_S}$  consists of all CQs that can be obtained as follows:

1. choose a CQ  $q(\mathbf{x})$  from  $q_{\Pi}$ , identify variables in  $q$  to obtain a CQ  $q'(\mathbf{x}')$ , and choose a partition  $q_1(\mathbf{x}_1), \dots, q_n(\mathbf{x}_n)$  of  $q'(\mathbf{x}')$ ;
2. for each  $i \in \{1, \dots, n\}$ , choose a relation  $R_{p(\mathbf{z})}$  from  $\mathbf{S}'_E$  and a vector  $\mathbf{y}$  of  $|\mathbf{z}|$  variables (repeated occurrences allowed) that are either from  $\mathbf{x}_i$  or do not occur in  $\mathbf{x}'$  such that  $q_i(\mathbf{x}_i) \subseteq p[\mathbf{y}/\mathbf{z}]$ ; then replace  $q_i(\mathbf{x}_i)$  in  $q'(\mathbf{x}')$  with the single atom  $R_{p(\mathbf{z})}(\mathbf{y})$ .

To establish that  $q_{\Pi_S}$  is as desired, we show that for every  $\mathbf{S}'_E$ -instance  $I'$

- (I)  $I' \models q_{\Pi_S}$  implies  $I' \models \Pi_S$  if  $I'$  is of girth exceeding  $k$  (soundness) and
- (II)  $I' \models \Pi_S$  implies  $I' \models q_{\Pi_S}$  (completeness).

Let  $I$  be the  $\mathbf{S}_E$ -instance corresponding to  $I'$ .

For Point (I), we observe that  $I' \models q_{\Pi_S}$  implies  $I \models q_{\Pi}$  (by construction of  $q_{\Pi_S}$  and of  $I'$ ) implies  $I \models \Pi$  (by choice of  $q_{\Pi}$ ) implies  $I' \models \Pi_S$  (by Point 2b of

Theorem 1 and if  $I'$  is of girth exceeding  $k$ ). Let us zoom into the first implication. Assume that  $I' \models q_{\Pi_S}$ . Then there is a CQ  $p_0(\mathbf{u})$  in  $q_{\Pi_S}$  and a homomorphism  $h$  from  $p_0(\mathbf{u})$  to  $I'$ . There must be some CQ  $q(\mathbf{x})$  in  $q_{\Pi}$  from which  $p_0(\mathbf{u})$  has been constructed in Steps 1 and 2 above. Let  $q_1(\mathbf{x}_1), \dots, q_n(\mathbf{x}_n)$  be as in this construction. It suffices to show that  $h$  is a homomorphism from  $q_i(\mathbf{x}_i)$  to  $I$ , for each  $i$ . Thus fix a  $q_i(\mathbf{x}_i)$ . Then there is a relation  $R_{p(\mathbf{z})} \in \mathbf{S}'_E$  and a vector  $\mathbf{y}$  of variables that are either from  $\mathbf{x}_i$  or do not occur in  $\mathbf{x}'$  such that  $q_i(\mathbf{x}_i) \subseteq p[\mathbf{y}/\mathbf{z}]$  and  $R_{p(\mathbf{z})}(\mathbf{y}) \in p_0(\mathbf{u})$ . Thus  $R_{p(\mathbf{z})}(h(\mathbf{y})) \in I'$ . By construction of  $I'$ , this yields  $q_i(h(\mathbf{x}_i)) \subseteq I$  and thus we are done.

For Point (II), we have that  $I' \models \Pi_S$  implies  $I \models \Pi$  (by Point 2a of Theorem 1) implies  $I \models q_{\Pi}$  (by choice of  $q_{\Pi}$ ). It thus remains to show that  $I \models q_{\Pi}$  implies  $I' \models q_{\Pi_S}$ . Thus assume that there is a CQ  $q(\mathbf{x})$  in  $q_{\Pi}$  and a homomorphism  $h$  from  $q(\mathbf{x})$  to  $I$ . We use  $q(\mathbf{x})$  and  $h$  to guide the choices in Step 1 and Step 2 of the construction of CQs in  $q_{\Pi_S}$  to exhibit a CQ  $p_0$  in  $q_{\Pi_S}$  such that  $p_0 \rightarrow I'$ .

We start with Step 1. As  $q'(\mathbf{x}')$ , we use the query obtained from  $q(\mathbf{x})$  by identifying variables  $x$  and  $y$  whenever  $h(x) = h(y)$ . Thus,  $h$  is an injective homomorphism from  $q'(\mathbf{x}')$  to  $I$ . We next need to choose a partition of  $q'(\mathbf{x}')$ . For every fact  $R(\mathbf{a}) \in I$ , choose a fact  $R_{p(\mathbf{x}_0)}(\mathbf{b}) \in I'$  that  $R(\mathbf{a})$  was obtained from during the construction of  $I$  and denote this fact with  $\mu(R(\mathbf{a}))$ . Now let  $q_1(\mathbf{x}_1), \dots, q_n(\mathbf{x}_n)$  be the partition of  $q'(\mathbf{x}')$  obtained by grouping together two atoms  $R_1(\mathbf{y}_1)$  and  $R_2(\mathbf{y}_2)$  if and only if  $\mu(R_1(h(\mathbf{y}_1))) = \mu(R_2(h(\mathbf{y}_2)))$ . Let  $\mu(q_i)$  denote the (unique) value of  $\mu$  for all the atoms in  $q_i(\mathbf{x}_i)$ .

Step 2 deals with each query  $q_i(\mathbf{x}_i)$  separately. We choose the relation  $R_{p(\mathbf{z})}$  from  $\mu(q_i) = R_{p(\mathbf{z})}(\mathbf{b})$ , which clearly is in  $\mathbf{S}'_E$ . We choose the vector  $\mathbf{y}$  of variables based on the vector of individuals  $\mathbf{b}$ . Let  $\mathbf{b} = b_1, \dots, b_n$ . Then the  $\ell$ -th variable in  $\mathbf{y}$  is  $y$  if  $h(y) = b_\ell$  (which is well-defined since  $h$  is injective) and a fresh variable if there is no such  $y$ . This finishes the guiding process and thus gives raise to a query  $p_0(\mathbf{u})$  in  $q_{\Pi_S}$ .

It remains to argue that  $h$  can be extended to a homomorphism  $h'$  from  $p(\mathbf{u})$  to  $I'$ . Take a  $q_i(\mathbf{x}_i)$  and consider the corresponding atom  $R_{p(\mathbf{z})}(\mathbf{y})$  in  $p_0$ . Then all the facts in  $q_i(h(\mathbf{x})) \subseteq I$  were obtained from the fact  $\mu(q_i) = R_{p(\mathbf{z})}(\mathbf{b}) \in I'$  during the construction of  $I$ . By construction of  $\mathbf{y}$  from  $\mathbf{b}$ , we can extend  $h$  to the fresh variables in  $\mathbf{y}$  so that  $h(\mathbf{y}) = \mathbf{b}$  and thus  $R_{p(\mathbf{z})}(h(\mathbf{y})) \in I'$ . Doing this for all  $q_i$  yields the desired  $h'$ .

Now for the cases  $\mathcal{Q} \in \{\text{MDLog}, \text{DLog}\}$ . We treat these cases in one since our construction preserves the width of Datalog-rewritings. In fact, this construction is very similar to the case  $\mathcal{Q} = \text{UCQ}$ , so we only give a sketch.

For Point 1, let  $\Gamma_{\Pi_S}$  be a Datalog-rewriting of  $\Pi_S$ . We construct a Datalog program  $\Gamma_{\Pi}$  of the same width over EDB schema  $\mathbf{S}_E$  by modifying the EDB part of each rule body in the same way in which we had modified the UCQ-rewriting  $q_{\Pi_S}$  in the case  $\mathcal{Q} = \text{UCQ}$ : replace every EDB-atom  $R_{q(\mathbf{x})}(\mathbf{y})$  with  $q[\mathbf{y}/\mathbf{x}]$ . We then have  $I \models \Pi$  iff  $I' \models \Pi_S$  (by Point 1 of Theorem 1) iff  $I' \models \Gamma_{\Pi_S}$  (by choice of  $\Gamma_{\Pi_S}$ ) iff  $I \models \Gamma_{\Pi}$ . The latter is by construction of  $I'$  and of  $\Gamma_{\Pi}$ . To prove it in more detail, it suffices to show that for every extension  $J$  of  $I$  to the IDB

relations in  $\Gamma_{\Pi_S}$  with corresponding extension  $J'$  of  $I'$ , and every rule body  $q$  in  $\Gamma_{\Pi_S}$  which was translated into a rule body  $q'$  in  $\Gamma_{\Pi}$ , we have  $q \rightarrow J$  iff  $q' \rightarrow J'$ . The arguments needed are as in the case  $\mathcal{Q} = \text{UCQ}$ .

The proof of Point 2 can be adapted from UCQs to Datalog in an analogous way. Let  $\Gamma_{\Pi}$  be a Datalog-rewriting of  $\Pi$ . We construct a Datalog program  $\Gamma_{\Pi_S}$  of the same width over EDB schema  $\mathbf{S}_{E'}$ . The rules in  $\Gamma_{\Pi_S}$  are obtained by taking a rule

$$P_0(\mathbf{x}_0) \leftarrow P_1(\mathbf{x}_1) \wedge \cdots \wedge P_n(\mathbf{x}_n) \wedge q(\mathbf{y})$$

from  $\Gamma_{\Pi}$ , where the  $P_i$  are IDB and  $q(\mathbf{y})$  is a CQ over schema  $\mathbf{S}_E$ , converting  $q(\mathbf{y})$  into a CQ  $q'(\mathbf{y}')$  over schema  $\mathbf{S}'_E$  in two steps, in the same way in which a CQ over  $\mathbf{S}_E$  was converted into a CQ over  $\mathbf{S}'_E$  in the case  $\mathcal{Q} = \text{UCQ}$ , and then including in  $\Gamma_{\Pi_S}$  the rule

$$P_0(\mathbf{x}_0) \leftarrow P_1(\mathbf{x}_1) \wedge \cdots \wedge P_n(\mathbf{x}_n) \wedge q'(\mathbf{y}').$$

The crucial step in the correctness proof is to show that  $I \models \Gamma_{\Pi}$  implies  $I' \models \Gamma_{\Pi_S}$  for any  $\mathbf{S}'_E$ -instance  $I'$  and corresponding  $\mathbf{S}_E$ -instance  $I$ . The arguments are again the same as in the case  $\mathcal{Q} = \text{UCQ}$ , the main difference being that we need to consider extensions of  $I$  and  $I'$  to IDB relations from  $\Gamma_{\Pi}$  instead of working with  $I$  and  $I'$  themselves.  $\square$

Point 2 of Lemma 1 only yields a rewriting of  $\Pi_S$  on  $\mathbf{S}'_E$ -instances of high girth. We next show that, for  $\mathcal{Q} \in \{\text{UCQ}, \text{MDLog}\}$ , the existence of a  $\mathcal{Q}$ -rewriting on high girth implies the existence of a  $\mathcal{Q}$ -rewriting that works on instances of unrestricted girth. Whether the same is true for  $\mathcal{Q} = \text{Datalog}$  remains as an open problem. We need the following well-known lemma that goes back to Erdős and was adapted to CSPs by Feder and Vardi. Informally, it says that every instance can be ‘exploded’ into an instance of high girth that behaves similarly regarding homomorphisms.

**Lemma 2.** *For every instance  $I$  and  $g, s \geq 0$ , there is an instance  $I'$  (over the same schema) such that  $I' \rightarrow I$ ,  $I'$  has girth exceeding  $g$ , and for every instance  $T$  of size at most  $s$ , we have  $I \rightarrow T$  iff  $I' \rightarrow T$ .*

Feder and Vardi additionally show that  $I'$  can be constructed by a randomized polynomial time reduction that was later derandomized by Kun [31], but here we do not rely on such computational properties. Every CQ  $q$  can be viewed as an instance  $I_q$  by using the variables as constants and the atoms as facts. It thus makes sense to speak about tree decompositions of CQs and about their treewidth, and it is clear what we mean by saying that a CQ is a tree (that is, has tree width  $(1, 1)$ ).

**Lemma 3.** *Let  $S$  be a set of templates over schema  $\mathbf{S}_E$ ,  $g \geq 0$ , and  $\mathcal{Q} \in \{\text{UCQ}, \text{MDLog}\}$ . If  $\text{coCSP}(S)$  is  $\mathcal{Q}$ -rewritable on instances of girth exceeding  $g$ , then it is  $\mathcal{Q}$ -rewritable.*

**Proof.** We start with  $\mathcal{Q} = \text{UCQ}$ . Let  $q_g$  be a UCQ that defines  $\text{coCSP}(S)$  on instances of girth exceeding  $g$ , and let  $q$  be the UCQ that consists of all tree

CQs which can be obtained from a CQ in  $q_g$  by identifying variables. We show that  $q$  defines  $\text{coCSP}(S)$  on unrestricted  $\mathbf{S}_E$ -instances.

Let  $I$  be an  $\mathbf{S}_E$ -instance. First assume that  $I \not\rightarrow S$ . By Lemma 2, there is an  $\mathbf{S}_E$ -instance  $I'$  of girth exceeding  $g$  and also exceeding the number of variables in each CQ in  $q_g$  and satisfying  $I' \rightarrow I$  and  $I' \not\rightarrow S$ . Thus  $I' \models q_g$ , that is, there is a CQ  $q'$  in  $q_g$  and a homomorphism  $h$  from  $q'$  to  $I'$ . Let  $q''$  be obtained from  $q'$  by identifying variables  $x$  and  $y$  if  $h(x) = h(y)$ . Thus,  $h$  is an injective homomorphism from  $q''$  to  $I'$ . Since the girth of  $I'$  exceeds the number of variables in  $q''$ ,  $q''$  must be a tree. Consequently,  $q''$  is a CQ in  $q$  and we have  $I' \models q$ . From  $I' \rightarrow I$ , we obtain  $I \models q$ .

Now assume that  $I \models q$ . Then, there is a tree CQ  $q'$  in  $q$  such that  $q' \rightarrow I$ . When we view  $q'$  as an  $\mathbf{S}_E$ -instance  $I_{q'}$ , then clearly  $I_{q'} \models q_g$  and  $I_{q'}$  has girth exceeding  $k$ . Thus,  $q' \not\rightarrow S$ , and from  $q' \rightarrow I$  we obtain  $I \not\rightarrow S$ .

Now for the case  $\mathcal{Q} = \text{MDLog}$ . Let  $\Gamma_g$  be an MDLog program that defines  $\text{coCSP}(S)$  on instances of girth exceeding  $g$ . Let  $\Gamma$  be the program obtained from  $\Gamma_g$  by replacing every rule  $P(x) \leftarrow q(\mathbf{x})$  with all rules  $P(x) \leftarrow q'(\mathbf{x}')$  such that  $q'(\mathbf{x}')$  is a tree CQ that can be obtained from  $q(\mathbf{x})$  by identifying variables. We show that  $\Gamma$  is an MDLog-definition of  $\text{coCSP}(S)$  on instances of unrestricted girth.

Let  $I$  be an  $\mathbf{S}_E$ -instance. First assume that  $I \not\rightarrow S$ . By Lemma 2, there is an  $\mathbf{S}_E$ -instance  $I'$  whose girth exceeds  $g$  and also exceeds the diameter of  $\Gamma_g$  and that satisfies  $I' \rightarrow I$  and  $I' \not\rightarrow S$ . The latter yields  $I' \models \Gamma_g$ . It remains to show that this implies  $I' \models \Gamma$  since with  $I' \rightarrow I$ , this yields  $I \models \Gamma$  as required.

To show that  $I' \models \Gamma$  follows from  $I' \models \Gamma_g$ , it suffices to show that all IDB facts derived by  $\Gamma_g$  starting from  $I'$  are also derived by  $\Gamma$ . Thus let  $J'$  be an extension of  $I'$  to the IDBs in  $\Gamma_g$ . It is enough to show that when a single application of a rule from  $\Gamma_g$  in  $J'$  yields an IDB atom  $P(a)$ , then  $\Gamma$  can derive the same atom. The former is the case only if  $\Gamma_g$  contains a rule  $P(x) \leftarrow q(\mathbf{x})$  such that there is a homomorphism  $h$  from  $q(\mathbf{x})$  to  $J'$  with  $h(x) = a$ . Let  $q'(\mathbf{x}')$  be the CQ obtained from  $q(\mathbf{x})$  by identifying variables  $x$  and  $y$  when  $h(x) = h(y)$ . Since the girth of  $I'$  exceeds the diameter of  $\Gamma_g$ ,  $q'(\mathbf{x}')$  is a tree. Thus,  $\Gamma$  contains the rule  $P(x) \leftarrow q'(\mathbf{x}')$  and the application of this rule in  $J'$  enabled by  $h$  yields  $P(a)$ . We have thus shown  $I' \models \Gamma$  and are done.

Now assume that  $I \models \Gamma_g$ . Then there is a proof tree for  $\text{goal}()$  from  $I$  and  $\Gamma_g$ , see [1] for details. From that tree, we can read off an  $\mathbf{S}_E$ -instance  $O$  such that  $O \rightarrow I$ ,  $O \models \Gamma_g$ , and, since  $\Gamma_g$  is monadic and only comprises rules with tree-shaped bodies,  $O$  is tree-shaped. Thus,  $O$  has girth exceeding  $g$  and from  $O \models \Gamma_g$  we get  $O \not\rightarrow S$ . But with  $O \rightarrow I$ , this yields  $I \not\rightarrow S$  as required.  $\square$

Putting together Theorem 1 and 2, Proposition 1, and Lemmas 1 and 3, we obtain the following reductions of rewritability of Boolean MDDLog programs to CSP rewritability.

**Proposition 2.** *Every Boolean MDDLog program  $\Pi$  can be converted into a set of templates  $S_\Pi$  such that*

1.  $\Pi$  is  $\mathcal{Q}$ -rewritable iff  $S_\Pi$  is  $\mathcal{Q}$ -rewritable for every  $\mathcal{Q} \in \{FO, UCQ, MDLog\}$ ;

2. every  $\mathcal{Q}$ -rewriting of  $\Pi$  can be effectively translated into a  $\mathcal{Q}$ -rewriting of  $S_\Pi$  and vice versa, for every  $\mathcal{Q} \in \{UCQ, MDLog\}$ .
3.  $|S_\Pi| \leq 2^{2^{p(n)}}$  and  $|T| \leq 2^{2^{p(n)}}$  for each  $T \in S_\Pi$ ,  $n$  the size of  $\Pi$  and  $p$  a polynomial.

The construction takes time polynomial in  $\sum_{T \in S_\Pi} |T|$ .

FO-rewritability of CSPs is NP-complete [32] and it was observed in [13] that the upper bound lifts to generalized CSPs. MDLog-rewritability of coCSPs is NP-hard and in EXPTIME [21]. We show in Appendix B that also this upper bound lifts to generalized coCSPs. Together with Proposition 2, this yields the upper bounds in the following theorem. The lower bounds are from [17].

**Theorem 3.** *For Boolean MDDLog programs and the complement of MMSNP sentences,*

1. *FO-rewritability (equivalently: UCQ-rewritability) is 2NEXPTIME-complete;*
2. *MDLog-rewritability is in 3EXPTIME (and 2NEXPTIME-hard).*

## 5 Shape of Rewritings, Obstructions, Explosion

In the FO case, it is possible to extract from the approach in the previous section an algorithm that computes actual rewritings, if they exist. However, that algorithm is hardly practical. An important first step towards the design of more practical algorithms that compute rewritings (in an exact or in an approximative way) is to analyze the shape of rewritings. In fact, both FO- and MDLog-rewritings of CSPs are known to be of a rather restricted shape, far from exploiting the full expressive power of the target languages. In this section, we establish corresponding results for Boolean MDDLog. This topic is closely related to the theory of obstructions, so we also establish connections between the rewritability of MMSNP sentences and natural obstruction sets. Finally, we observe an MMSNP counterpart of the fundamental ‘explosion’ lemma for CSPs (Lemma 2).

The following summarizes our results regarding the shape of rewritings.

**Theorem 4.** *Let  $\Pi$  be a Boolean MDDLog program of diameter  $k$ . Then*

1. *if  $\Pi$  is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth  $(1, k)$ ;*
2. *if  $\Pi$  is MDLog-rewritable, then it has an MDLog-rewriting of diameter  $k$ .*

**Proof.** We analyze the proof of Lemma 1 and use known results from CSP. In fact, any FO-rewritable CSP has a UCQ-rewriting that consists of tree CQs, and thus the same holds for simple Boolean MDDLog programs. If we convert such a rewriting of  $\Pi_S$  into a rewriting of  $\Pi$  as in the proof of Lemma 1, we obtain a UCQ-rewriting in which each CQ has treewidth  $(1, k)$ . For Point 2 of Theorem 4, one uses the known fact that every MDLog-rewritable CSP has an MDLog-rewriting in which every rule body comprises at most one EDB atom, see e.g. proof of Theorem 19 in [26].  $\square$

In a sense, the concrete bound  $k$  in Points 1 and 2 of Theorem 4 is quite remarkable. Point 2 says, for example, that when eliminating disjunction from a Boolean MDDLog program, it never pays off to increase the diameter.

We now consider obstructions. An *obstruction set*  $\mathcal{O}$  for a CSP template  $T$  over schema  $\mathbf{S}_E$  is a set of instances over the same schema such that for any  $\mathbf{S}_E$ -instance  $I$ , we have  $I \not\models T$  iff  $O \rightarrow I$  for some  $O \in \mathcal{O}$ . The elements of  $\mathcal{O}$  are called *obstructions*. A lot is known about CSP obstructions. For example,  $T$  is FO-rewritable if and only if it has a finite obstruction set [4] if and only if it has a finite obstruction set that consists of finite trees [38], and  $T$  is MDLog-rewritable if and only if it has a (potentially infinite) obstruction set that consists of finite trees [26]. Here we consider obstruction sets for MMSNP, defined in the obvious way: an *obstruction set*  $\mathcal{O}$  for an MMSNP sentence  $\theta$  over schema  $\mathbf{S}_E$  is a set of instances over the same schema such that for any  $\mathbf{S}_E$ -instance  $I$ , we have  $I \not\models \theta$  iff  $O \rightarrow I$  for some  $O \in \mathcal{O}$ . This should not be confused with colored forbidden patterns used to characterize MMSNP in [36]. The following characterizes FO-rewritability of MMSNP sentences in terms of obstruction sets.

**Corollary 1.** *For every MMSNP sentence  $\theta$ , the following are equivalent:*

1.  $\theta$  is FO-rewritable;
2.  $\theta$  has a finite obstruction set;
3.  $\theta$  has a finite set of finite obstructions of treewidth  $(1, k)$ .

Corollary 1 follows from Point 1 of Theorem 4 and the straightforward observations that an MMSNP sentence  $\theta$  is FO-rewritable iff  $\neg\theta$  is (which is equivalent to a Boolean MDDLog program) and that every finite obstruction set  $\mathcal{O}$  for  $\theta$  gives rise to a UCQ-rewriting  $\bigvee \mathcal{O}$  of  $\neg\theta$  and vice versa. We now turn to MDLog-rewritability.

**Proposition 3.** *Let  $\theta$  be an MMSNP sentence of diameter  $k$ . Then  $\neg\theta$  is MDLog-rewritable iff  $\theta$  has a set of obstructions (equivalently: finite obstructions) that are of treewidth  $(1, k)$ .*

**Proof.** The “only if” direction is a consequence of Point 2 of Theorem 4 and the fact that, for any Boolean monadic Datalog program  $\Pi \equiv \neg\theta$  of diameter  $k$  over EDB schema  $\mathbf{S}_E$ , a proof tree for  $\text{goal}()$  from an  $\mathbf{S}_E$ -instance  $I$  and  $\Pi$  gives rise to a finite  $\mathbf{S}_E$ -instance  $J$  of treewidth  $(1, k)$  with  $J \rightarrow I$ . The desired obstruction set for  $\neg\theta$  is then the set of all these  $J$ . The “if” direction is a consequence of Theorem 5 in [15].  $\square$

We remark the results in [15] almost give Proposition 3, but do not seem to deliver any concrete bound on the parameter  $k$  of the treewidth of obstruction sets.

We close with observing an MMSNP counterpart of Lemma 2, first giving a preliminary. Let  $I$  be an instance over some schema  $\mathbf{S}_E$ . A  $(1, k)$ -*decomposition* of  $I$  is a pair  $(V, (I_v)_{v \in V})$  where  $V$  is a set of indices and  $(I_v)_{v \in V}$  is a partition of  $I$  such that for all  $v, v' \in V$ ,  $|\text{dom}(I_v) \cap \text{dom}(I_{v'})| \leq 1$  and  $|\text{dom}(I_v)| \leq k$ . Thus, a  $(1, k)$ -decomposition  $D = (V, (I_v)_{v \in V})$  decomposes  $I$  into parts of size at most

$k$  and with little overlap. These parts can be viewed as the facts of an instance  $I_D$  over an aggregation schema  $\mathbf{S}'_E$  defined by the relations  $R_{q_v(\mathbf{x})}$  where  $q_v(\mathbf{x})$  is  $I_v$  viewed as a CQ, that is,

$$I_D = \{R_{q_v(x)}(\text{dom}(I_v)) \mid v \in V\}$$

where we assume some fixed (but otherwise irrelevant) order on the elements of each  $\text{dom}(I_v)$ . Now, we say that  $I$  has  $(1, k)$ -decomposition girth  $g$  if  $g$  is the maximum girth of  $I_D$  for any  $(1, k)$ -decomposition  $D$  of  $I$ . It can be shown that the  $(1, k)$ -decomposition girth of  $I$  is undefined (which intuitively corresponds to  $I$  having infinite  $(1, k)$ -decomposition girth) if and only if it has treewidth  $(1, k)$ . Here comes the announced MMSNP counterpart of Lemma 2.

**Lemma 4.** *For every instance  $I$  and  $g \geq s > 0$ , and every MDDLog program  $\Pi$  of diameter at most  $s$ , there is an instance  $J$  (over the same schema) such that  $J \rightarrow I$ ,  $J$  has  $(1, s)$ -decomposition girth exceeding  $g$ , and  $I \models \Pi$  iff  $J \models \Pi$ .*

**Proof.** Let  $\Pi$  be a Boolean MDDLog program of diameter  $k \leq s$  over EDB schema  $\mathbf{S}_E$ . By Theorems 1 and 2, there is a  $k$ -aggregation schema  $\mathbf{S}'_E$  and a set of templates  $\mathbf{S}_\Pi$  over  $\mathbf{S}'_E$  such that

1. for any  $\mathbf{S}_E$ -instance  $I$  with corresponding  $\mathbf{S}'_E$ -instance  $I'$ ,  $I \models \Pi$  iff  $I' \not\models S_\Pi$ ;
2. for any  $\mathbf{S}'_E$ -instance  $I'$  whose girth exceeds  $k$  with corresponding  $\mathbf{S}_E$ -instance  $I$ ,  $I' \not\models S_\Pi$  iff  $I \models \Pi$ .

Let  $I$  and  $I'$  be an  $\mathbf{S}_E$ -instance and its corresponding  $\mathbf{S}'_E$ -instance. Furthermore, let  $J'$  be the  $\mathbf{S}'_E$ -instance obtained from  $I'$  by applying Lemma 2 with  $s = \max\{|T| \mid T \in S_\Pi\}$  and  $g$  as given. Then  $J' \not\models I'$ ,  $J'$  has girth exceeding  $g$ , and  $J' \not\models S_\Pi$  iff  $I' \not\models S_\Pi$  iff  $I \models \Pi$ . Let  $J$  be the  $\mathbf{S}_E$ -instance corresponding to  $J'$ . As  $J'$  has girth exceeding  $k$ ,  $J \models \Pi$  iff  $J' \not\models S_\Pi$  iff  $I \models \Pi$ .

It remains to show that  $J$  has  $(1, s)$ -decomposition girth exceeding  $g$  and that  $J \rightarrow I$ . The former is witnessed by the  $(1, k)$ -decomposition  $D = (V, (I_v)_{v \in V})$  of  $J$  obtained by using as  $V$  the facts of  $J'$  and as  $I_v$  the set of facts obtained from  $v$  during the construction of  $J$ .

As the last step, we argue that  $J \rightarrow I$  follows from  $J' \rightarrow I'$ , and that in fact any homomorphism  $h$  from  $J'$  to  $I'$  is also a homomorphism from  $J$  to  $I$ . Thus let  $h$  be such a homomorphism. For any fact  $R(a_{i_1}, \dots, a_{i_\ell})$  in  $J$ , there is a fact  $R_{q(x_1, \dots, x_n)}(a_1, \dots, a_m)$  in  $J'$  such that  $R(x_{i_1}, \dots, x_{i_\ell}) \in q_i(x_1, \dots, x_n)$ . We have  $R_{q(x_1, \dots, x_n)}(h(a_1), \dots, h(a_m)) \in I'$ . By definition of  $I'$ , this means  $R(h(a_{i_1}), \dots, h(a_{i_\ell})) \in I$  and we are done.  $\square$

We believe that Lemma 4 can be useful in many contexts, saving a detour via CSPs. For example, it enables an alternative proof of Theorem 4. We illustrate this for Point 1. We can start with a UCQ-rewriting  $q$  of an MDDLog program  $\Pi$  of diameter  $k$  and show that the UCQ  $q_t$  that consists of all CQs of treewidth  $(1, k)$  that can be obtained from a CQ in  $q$  by identifying variables must also be a rewriting of  $\Pi$ : take an instance  $I$  that makes  $\Pi$  true, use Lemma 4 to transform  $I$  to an  $I'$  of girth exceeding  $k$  and also exceeding the size of any CQ in  $q$  such



that  $I' \models \Pi$  and  $I' \rightarrow I$ , observe that  $I' \models q$  and that a homomorphism from any CQ  $p$  in  $q$  to  $I'$  gives rise to a homomorphism from a CQ  $p'$  in  $q_t$  to  $I'$ , and derive  $p' \rightarrow I$  from  $I' \rightarrow I$ .

## 6 Datalog-Rewritability of Boolean MDDLog Programs and Canonical Datalog Programs

We consider the rewritability of Boolean MDDLog programs into Datalog programs. In contrast to the case of rewriting into FO or into monadic Datalog, here we only obtain a procedure that is sound, but whose completeness remains an open problem. However, the procedure is complete for the class of MDDLog programs that have equality, a condition that is defined in detail below. We also give a new and direct construction of canonical Datalog-rewritings of Boolean MDDLog programs (equivalently: the complements of MMSNP sentences), bypassing the construction of infinite templates which involves the application of a non-trivial construction due to Cherlin, Shelah, and Shi [15, 22].

### 6.1 Datalog-Rewritability of Boolean MDDLog Programs

We say that an MDDLog program  $\Pi$  *has equality* if its EDB schema includes the distinguished binary relation  $\text{eq}$ ,  $\Pi$  contains the rules

$$P(x) \wedge \text{eq}(x, y) \rightarrow P(y) \quad \text{and} \quad P(y) \wedge \text{eq}(x, y) \rightarrow P(x)$$

for each IDB relation  $P$ , and these are the only rules that mention  $\text{eq}$ . For an MDDLog program  $\Pi$  that does not have equality, we use  $\Pi^=$  to denote the extension of  $\Pi$  with the fresh EDB relation  $\text{eq}$  and the above rules. If  $\Pi$  has equality, then  $\Pi^=$  simply denotes  $\Pi$ .

**Lemma 5.** *For any MDDLog program  $\Pi$ , DLog-rewritability of  $\Pi^=$  implies DLog-rewritability of  $\Pi$ .*

Lemma 5 follows from the trivial observation that any DLog-rewriting of  $\Pi^=$  can be converted into a DLog-rewriting of  $\Pi$  by dropping all rules that use the relation  $\text{eq}$ . It remains an interesting open question whether the converse of Lemma 5 holds.

We shall also need a notion of a CSP to have equality, which is actually more natural than in the MMSNP case. A CSP template  $T$  *has equality* if its EDB schema includes the distinguished binary relation  $\text{eq}$  and  $T$  interprets  $\text{eq}$  as the relation  $\{(a, a) \mid a \in \text{dom}(T)\}$ . It can be verified that when an MDDLog program that has equality is converted into a generalized CSP based on a set of templates  $S_\Pi$  according to Theorems 1 and 2 (using the concrete constructions in the appendix), then all templates in  $S_\Pi$  have equality. The interesting aspect of having equality is that it allows us to establish a counterpart of Lemma 3 also for Datalog-rewritability.

**Lemma 6.** *Let  $S$  be a set of templates over schema  $\mathbf{S}_E$  that have equality, and let  $g \geq 0$ . If  $\text{coCSP}(S)$  is DLog-rewritable on instances of girth exceeding  $g$ , then it is DLog-rewritable.*

**Proof.** With every  $\mathbf{S}_E$ -instance  $I$  and  $g \geq 0$ , we associate an  $\mathbf{S}_E$ -instance  $I^g$  of girth exceeding  $g$  such that for any template  $T$  over  $\mathbf{S}_E$  that has equality,  $I^g \rightarrow T$  iff  $I \rightarrow T$ .

Thus, let  $I$  be an  $\mathbf{S}_E$ -instance and  $g \geq 0$ . Let  $\text{pos}(I)$  be the set of pairs  $(R(\mathbf{a}), i)$  such that  $R(\mathbf{a}) \in I$  and  $i \in \text{pos}(R)$ . Reserve fresh constants as follows:

- a constant  $b_p$ , for all  $p = (R(\mathbf{a}), i) \in \text{pos}(I)$ ;
- constants  $b_{p,p',1}, \dots, b_{p,p',g}$ , for all  $p, p' = (R(\mathbf{a}), i), (R'(\mathbf{a}'), i') \in \text{pos}(I)$  with  $a_i = a'_{i'}$ .

The instance  $I^g$  then consists of the following facts:

1. for every  $R(\mathbf{a}) \in I$  with  $R$  of arity  $n$ , the fact  $R(b_{p_1}, \dots, b_{p_n})$  where  $p_i = (R(\mathbf{a}), i)$ ;
2. for all distinct  $p, p' = (R(\mathbf{a}), i), (R'(\mathbf{a}'), i') \in \text{pos}(I)$  with  $a_i = a'_{i'}$ , the facts

$$\text{eq}(b_p, b_{p,p',1}), \text{eq}(b_{p,p',1}, b_{p,p',2}), \dots, \text{eq}(b_{p,p',g-1}, b_{p,p',g}), \text{eq}(b_{p,p',g}, b_{p'}).$$

The resulting  $\mathbf{S}_E$ -instance  $I^g$  clearly has girth exceeding  $g$ . Moreover, for any template  $T$  over  $\mathbf{S}_E$  that has equality, there is a homomorphism  $h$  from  $I$  to  $T$  iff there is a homomorphism  $h_g$  from  $I^g$  to  $T$ . In fact,  $h_g$  can be obtained from  $h$  by setting  $h_g(b_p) = h_g(b_{p,p',j}) = h(a_i)$  when  $p = (R(\mathbf{a}), i)$ ; conversely,  $h$  can be obtained from  $h_g$  by setting  $h(a_i) = h_g(b_p)$  when  $p = (R(\mathbf{a}), i)$ —the latter is well-defined by construction of  $I^g$  and since  $\text{eq}$  is interpreted as the reflexive relation in  $T$ .

Now assume that  $\text{coCSP}(S)$  is DLog-rewritable on instances of girth exceeding  $g$  and let  $\Gamma$  be a concrete rewriting. We construct a Datalog program  $\Gamma'$  such that for any  $\mathbf{S}_E$ -instance  $I$ ,  $I \models \Gamma'$  iff  $I^g \models \Gamma$ . Clearly,  $\Gamma'$  is then a rewriting of  $\text{coCSP}(S)$  on instances of unrestricted girth.

We aim to construct  $\Gamma'$  such that it mimics the execution of  $\Gamma$  on  $I^g$ , despite being executed on  $I$ . One challenge is that the domains of  $I$  and  $I^g$  are not identical. In  $\Gamma'$ , the IDB relations of  $\Gamma$  need to be adapted to reflect this change of domain, and so do the rules. Let  $m$  be the maximum arity of any relation in  $\mathbf{S}_E$ . Every IDB relation  $P$  of  $\Gamma$  gives rise to a *set* of IDB relations in  $\Gamma'$ . In fact, every position of  $P$  can be replaced either with

1.  $\ell$  positions, for some  $\ell \leq m$ , reflecting the case that the position is filled with a constant  $b_p$ ; or with
2.  $\ell_1 + \ell_2$  positions, for some  $\ell_1, \ell_2 \leq m$ , reflecting the case that the position is filled with a constant  $b_{p,p',j}$ .

In Case 1, the  $\ell$  positions identify the constants in the fact  $R(\mathbf{a})$  in  $p$ . The symbol  $R$  and the number  $i$  from  $p$  also need to be stored, which is done as an annotation to the IDB relation. In Case 2, the first  $\ell_1$  positions identify the constants in

the fact  $R(\mathbf{a})$  in  $p$  while the latter  $\ell_2$  positions identify the constants in the fact  $R'(\mathbf{a}')$  in  $p'$ ; we additionally need to store the symbols  $R$  and  $R'$ , the numbers  $i$  and  $i'$  from  $p$  and  $p'$ , and the number  $j$ , which is again done by annotation of the IDB relation.

Let us make this formal. The IDB relations of  $\Gamma'$  take the form  $P^\mu$  where  $P$  is an IDB relation of  $\Gamma$  and  $\mu$  is a function from  $\text{pos}(P)$  to

$$\Omega := (\mathbf{S}_E \times \{1, \dots, m\}) \cup (\mathbf{S}_E \times \{1, \dots, m\} \times \mathbf{S}_E \times \{1, \dots, m\} \times \{1, \dots, g\})$$

such that if  $\mu(\ell) = (R, i)$ , then  $i \in \text{pos}(R)$  and if  $\mu(\ell) = (R, i, R', i', j)$ , then  $i \in \text{pos}(R)$  and  $i' \in \text{pos}(R')$ . The arity of  $P^\mu$  is  $\sum_{\ell=1..|\text{pos}(P)|} q_\ell$  where  $q_\ell$  is the arity of  $R$  if  $\mu(\ell) = (R, i)$  and  $q_\ell$  is the sum of the arities of  $R$  and  $R'$  if  $\mu(\ell) = (R, i, R', i', j)$ . In the construction of  $\Gamma'$ , we manipulate the rules of  $\Gamma$  to account for this change in the IDB schema. We can assume w.l.o.g. that  $\Gamma$  is closed under identifying variables in rules. Let

$$\begin{aligned} P_0(\mathbf{x}_0) \leftarrow & P_1(\mathbf{x}_1) \wedge \dots \wedge P_{\ell_1}(\mathbf{x}_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \dots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & \text{eq}(z_{1,1}, z_{1,2}) \wedge \dots \wedge \text{eq}(z_{\ell_3,1}, z_{\ell_3,2}) \end{aligned}$$

be a rule in  $\Gamma$  where  $P_0, \dots, P_{\ell_1}$  are IDB and  $R_1, \dots, R_{\ell_2}$  are EDB (possibly the distinguished  $\text{eq}$  relation), such that

(\*) every variable occurs at most once in  $R_1(\mathbf{y}_1) \wedge \dots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2})$ .

Note that it might be possible to write a single rule from  $\Gamma$  in the above form in more than one way; we then consider all possible ways. Also note that rules that do not satisfy (\*) can be ignored since they never apply in  $I^g$ .

Let  $\mathbf{x}$  be the variables in the rule, and let  $\delta : \mathbf{x} \rightarrow \Omega$  be such that the following conditions are satisfied:

1. for each  $R_i(\mathbf{y}_i)$  with  $\mathbf{y}_i = y_1 \dots y_k$ , we have  $\delta(y_j) = (R_i, j)$  for all  $j$ ;
2. for each  $\text{eq}(z_{i,1}, z_{i,2})$ , one of the following is true for some  $R, i, R', i', j$ :
  - (a)  $\delta(z_{i,1}) = (R, i)$  and  $\delta(z_{i,2}) = (R, i, R', i', 1)$ ;
  - (b)  $\delta(z_{i,1}) = (R, i, R', i', g)$  and  $\delta(z_{i,2}) = (R', i')$ ;
  - (c)  $\delta(z_{i,1}) = (R, i, R', i', j)$  and  $\delta(z_{i,2}) = (R, i, R', i', j + 1)$ ;
  - (d)  $\delta(z_{i,1}) = (R, i, R', i', j)$  and  $\delta(z_{i,2}) = (R, i, R', i', j - 1)$ .

With each variable  $x$  in  $\mathbf{x}$ , we associate a vector  $\mathbf{u}_x$  of distinct variables. If  $\delta(x)$  is of the form  $(R, i)$ , then the length of  $\mathbf{u}_x$  matches the arity of  $R$  and  $\mathbf{u}_x$  is called a *variable block*. If  $\delta(x)$  is of the form  $(R, i, R', i', j)$ , then the length of  $\mathbf{u}_x$  is the sum of the arities  $n$  and  $n'$  of  $R$  and  $R'$ ; the first  $n$  variables in  $\mathbf{u}_x$  are then also called a *variable block*, and so are the last  $n'$  variables. Variable blocks will either be disjoint or identical. Identities are minimized such that the following conditions are satisfied:

(I1) if  $x$  occurs in some  $\mathbf{y}_i$ , then  $\mathbf{u}_x = \mathbf{y}_i$ ;

- (I2) if Case 2a applies to  $\text{eq}(z_{i,1}, z_{i,2})$ , then  $\mathbf{u}_{z_{i,1}}$  is identical to the first variable block in  $\mathbf{u}_{z_{i,2}}$ ;
- (I3) if Case 2b applies to  $\text{eq}(z_{i,1}, z_{i,2})$ , then  $\mathbf{u}_{z_{i,2}}$  is identical to the second variable block in  $\mathbf{u}_{z_{i,1}}$ ;
- (I4) if Case 2c or 2d applies to  $\text{eq}(z_{i,1}, z_{i,2})$ , then the first variable blocks of  $\mathbf{u}_{z_{i,1}}$  and  $\mathbf{u}_{z_{i,2}}$  are identical, and so are the second variable blocks.

Regarding (I1), note that  $x$  cannot occur in more than one  $\mathbf{y}_i$  because of  $(*)$ , thus the condition can always be satisfied ‘without conflicts’. Then include in  $\Gamma'$  the rule

$$\begin{aligned} P_0^{\mu_0}(\mathbf{x}'_0) \leftarrow & P_1^{\mu_1}(\mathbf{x}'_1) \wedge \cdots \wedge P_{\ell_1}^{\mu_{\ell_1}}(\mathbf{x}'_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & W \end{aligned}$$

such that

- (R1) if the  $k$ -th component in  $\mathbf{x}_i$  is  $x$ , then  $\mu_i(k) = \delta(x)$ ;
- (R2)  $\mathbf{x}'_i$  is obtained from  $\mathbf{x}_i$  by replacing each variable  $x$  with  $\mathbf{u}_x$ ;
- (R3)  $W$  contains the following atoms:
  - for each variable  $x \in \mathbf{x}$  with  $\delta(x)$  of the form  $(R, i)$ , an atom  $R(\mathbf{w})$  where the  $i$ -th component of  $\mathbf{w}$  is  $x$  and all other variables are distinct and fresh;
  - for each variable  $x \in \mathbf{x}$  with  $\delta(x)$  of the form  $(R, i, R', i', j)$ , atoms  $R(\mathbf{w}), R'(\mathbf{w}')$  where the  $i$ -th component of  $\mathbf{w}$  and the  $j$ -th component of  $\mathbf{w}'$  is  $x$  and all other variables are distinct and fresh.

As an example, consider the following rule in  $\Gamma$ :

$$\text{goal}() \leftarrow R(x_1, x_2) \wedge P(x_1, x_3) \wedge \text{eq}(x_2, x_3)$$

where  $R$  is EDB and  $P$  IDB, and let  $\delta(x_1) = (R, 1)$ ,  $\delta(x_2) = (R, 2)$ , and  $\delta(x_3) = (R, 2, R, 1, 1)$ . Note that Case 2a applies to  $\text{eq}(x_2, x_3)$ . We have  $\mathbf{u}_{x_1} = \mathbf{u}_{x_2} = x_1x_2$  and  $\mathbf{u}_{x_3} = x_1x_2u_1u_3$  and thus obtain the following rule in  $\Gamma'$ :

$$\begin{aligned} \text{goal}() \leftarrow & R(x_1, x_2) \wedge P^\mu(x_1, x_2, x_1, x_2, u_1, u_2) \wedge \\ & R(x_1, w_1) \wedge R(x_2, w_2) \wedge R(z_3, x_3) \wedge R(x_3, z_4) \end{aligned}$$

where the last line corresponds to  $W$  above, and where  $\mu(1) = (R, 1)$  and  $\mu(2) = (R, 2, R, 1, 1)$ .

We have to show that  $I \models \Gamma'$  iff  $I^g \models \Gamma$  for any  $\mathbf{S}_E$ -instance  $I$ . There is a correspondence between extensions of  $I^g$  to the IDB relations in  $\Gamma$  and extensions of  $I$  to the IDB relations in  $\Gamma'$ . More precisely, a fact  $P^\mu(\mathbf{a})$  in an extension of  $I$  represents a fact  $P(\mathbf{b})$  in an extension of  $I^g$  as follows (and vice versa): for each  $i \in \text{pos}(P)$ , let  $\mathbf{a}_i$  be the subvector of  $\mathbf{a}$  that starts at position  $\sum_{\ell=1..i-1} q_\ell$  and is of length  $q_i$  (where, as before,  $q_\ell$  is the arity of  $R$  if  $\mu(\ell) = (R, i)$  and  $q_\ell$  is the sum of the arities of  $R$  and  $R'$  if  $\mu(\ell) = (R, i, R', i', j)$ ); the  $i$ -th constant in  $\mathbf{b}$  is  $b_{R(\mathbf{a}_i), j}$  if  $\mu(i) = (R, j)$  and  $b_{R(\mathbf{c}), j, R'(\mathbf{c}'), j', \ell}$  if  $\mu(i) = (R, j, R', j', \ell)$  and  $\mathbf{a}_i = \mathbf{c}\mathbf{c}'$ .

One essentially has to show that every application of a rule from  $\Gamma'$  in an extension of  $I$  can be reproduced by an application of a rule from  $\Gamma$  in the corresponding extension of  $I^g$ , and vice versa. We only sketch the details. First let  $J^g$  be an extension of  $I^g$  to the IDB relations in  $\Gamma$  and let  $P(\mathbf{y}) \leftarrow q(\mathbf{x})$  be a rule in  $\Gamma$  applicable in  $J^g$ , and  $h$  a homomorphism from  $q(\mathbf{x})$  to  $J^g$  such that  $P(h(\mathbf{y})) \notin J^g$ . Since  $\Gamma$  is closed under identifying variables in rules, we can assume that  $h$  is injective. Let

$$\begin{aligned} q(\mathbf{x}) = & P_1(\mathbf{x}_1) \wedge \cdots \wedge P_{\ell_1}(\mathbf{x}_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & \text{eq}(z_{1,1}, z_{1,2}) \wedge \cdots \wedge \text{eq}(z_{\ell_3,1}, z_{\ell_3,2}) \end{aligned}$$

such that all  $P_i$  are IDB, all  $R_i$  EDB, and an equality atom  $\text{eq}(x, y)$  is included in the third line if and only if at least one of  $h(x)$  and  $h(y)$  is not of the form  $b_p$ . Consequently, for all variables  $x$  that occur in the second line,  $h(x)$  is of the form  $b_p$ . One can now verify that Condition (\*) is satisfied. Assume that this is not the case. The first case is that there are distinct atoms  $R_i(\mathbf{y}_i)$  and  $R_j(\mathbf{y}_j)$  that share a variable  $x$ . In  $I^g$ , every constant of the form  $b_p$  occurs in exactly one fact that only contains constants of the form  $b_p$ . Thus,  $h$  must take  $R_i(\mathbf{y}_i)$  and  $R_j(\mathbf{y}_j)$  to the same fact in  $J^g$ . Since  $h$  is injective,  $R_i(\mathbf{y}_i)$  and  $R_j(\mathbf{y}_j)$  must be identical which is a contradiction. The second case is that there is an atom  $R_i(\mathbf{y}_i)$  in which a variable occurs more than once. This is in contradiction to  $h$  being a homomorphism to  $J^g$ .

Now define a map  $\delta : \mathbf{x} \rightarrow \Omega$  by putting  $\delta(x) = p$  if  $h(x) = b_p$  and  $\delta(x) = (p, p', i)$  if  $h(x) = b_{p,p',i}$ . It can be verified that the two conditions required of  $\delta$  are satisfied. We thus obtain a corresponding rule in  $\Gamma'$ . It can be verified that applying this rule in the extension  $J$  of  $I$  corresponding to  $J^g$  adds the fact that corresponds to  $P(h(\mathbf{y}))$ .

Conversely, let  $J$  be an extension of  $I$  to the IDB relations in  $\Gamma'$  and let

$$\begin{aligned} P_0^{\mu_0}(\mathbf{x}'_0) \leftarrow & P_1^{\mu_1}(\mathbf{x}'_1) \wedge \cdots \wedge P_{\ell_1}^{\mu_{\ell_1}}(\mathbf{x}'_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & W \end{aligned}$$

be a rule in  $\Gamma'$  and  $h$  a homomorphism from the rule body to  $J$  such that  $P^{\mu_0}(h(\mathbf{x}'_0))$  is not in  $J$ . This rule was derived from a rule

$$\begin{aligned} P_0(\mathbf{x}_0) \leftarrow & P_1(\mathbf{x}_1) \wedge \cdots \wedge P_{\ell_1}(\mathbf{x}_{\ell_1}) \wedge \\ & R_1(\mathbf{y}_1) \wedge \cdots \wedge R_{\ell_2}(\mathbf{y}_{\ell_2}) \wedge \\ & \text{eq}(z_{1,1}, z_{1,2}) \wedge \cdots \wedge \text{eq}(z_{\ell_3,1}, z_{\ell_3,2}) \end{aligned}$$

in  $\Gamma$  and a map  $\delta : \mathbf{x} \rightarrow \Omega$ ,  $\mathbf{x}$  the variables in the latter rule. We define a map  $h'$  from  $\mathbf{x}$  to  $\text{dom}(J^g)$ , where  $J^g$  is the extension of  $I^g$  that corresponds to  $J$ . Let  $x \in \mathbf{x}$ . If  $\delta(x) = (R, i)$  and  $h(\mathbf{u}_x) = \mathbf{a}$ , then set  $h'(x) = b_{R(\mathbf{a}),i}$ . If  $\delta(x) = (R, i, R', i', j)$  and  $h(\mathbf{u}_x) = \mathbf{a}\mathbf{a}'$ , then set  $h'(x) = b_{R(\mathbf{a}),i,R'(\mathbf{a}'),i',j}$ . We argue that  $h'$  is a homomorphism from the body of the latter rule to  $J^g$ . There are three cases:

- Consider an atom  $P_i(\mathbf{x}_i)$ . Let  $\mathbf{x}_i = x_1 \cdots x_n$ . Then there is a corresponding atom  $P_i^{\mu_i}(\mathbf{x}_i)$  in the former rule and thus  $P_i^{\mu_i}(h(\mathbf{x}_i)) \in J$ . For each  $j \in \text{pos}(P_i)$ , let  $\mathbf{a}_j$  be the subvector of  $h(\mathbf{x}_i)$  that starts at position  $\sum_{\ell=1..j-1} q_\ell$  and is of length  $q_j$ . Define the vector  $\mathbf{b}$  by letting the  $j$ -th constant be  $b_{R(\mathbf{a}_j), \ell}$  if  $\mu(j) = (R, \ell)$  and  $b_{R(\mathbf{c}), \ell, R'(\mathbf{c}'), \ell', k}$  if  $\mu(j) = (R, \ell, R', \ell', k)$  and  $\mathbf{a}_i = \mathbf{c}\mathbf{c}'$ . By (R3), all constants in  $\mathbf{b}$  occur in the domain of  $J^g$ . Moreover,  $P_i(\mathbf{b}) \in J^g$ . It thus remains to observe that  $h'(\mathbf{x}_i) = \mathbf{b}$ , which follows from (R1) and (R2) and the definition of  $h'$ .  
implies  $R(b_{R_i(h(\mathbf{y}_i)), 1}, \dots, b_{R_i(h(\mathbf{y}_i)), n}) \in J^g$ . By Condition 1 imposed on  $\delta$ , we have  $\delta(y_j) = (R_i, j)$  for each  $j$ . Moreover, by (I1) we must have  $\mathbf{u}_{y_j} = \mathbf{y}_i$  for each  $j$ . Thus, the definition of  $h'$  yields  $h'(\mathbf{y}_i) = b_{R_i(h(\mathbf{y}_i)), 1} \cdots b_{R_i(h(\mathbf{y}_i)), n}$  and we are done.
- Consider an atom  $R_i(\mathbf{y}_i)$ . Let  $\mathbf{y}_i = y_1 \cdots y_n$ . Then the atom  $R_i(\mathbf{y}_i)$  must also be in the former rule and thus  $R_i(h(\mathbf{y}_i)) \in J$ , from which we obtain  $R(b_{R_i(h(\mathbf{y}_i)), 1}, \dots, b_{R_i(h(\mathbf{y}_i)), n}) \in J^g$ . By Condition 1 imposed on  $\delta$ , we have  $\delta(y_j) = (R_i, j)$  for each  $j$ . Moreover, by (I1) we must have  $\mathbf{u}_{y_j} = \mathbf{y}_i$  for each  $j$ . Thus, the definition of  $h'$  yields  $h'(\mathbf{y}_i) = b_{R_i(h(\mathbf{y}_i)), 1} \cdots b_{R_i(h(\mathbf{y}_i)), n}$  and we are done.
- Consider an atom  $\text{eq}(z_{i,1}, z_{i,2})$ . We know that one of the Cases 2a to 2d apply to  $\text{eq}(z_{i,1}, z_{i,2})$ . We only treat the first case explicitly. Thus assume that  $\delta(z_{i,1}) = (R, j)$  and  $\delta(z_{i,2}) = (R, j, R', j', 1)$ . By definition,  $h'(z_{i,1}) = b_{R(h(\mathbf{u}_{z_{i,1}})), j}$  and  $h'(z_{i,2}) = b_{R(\mathbf{c}), j, R'(\mathbf{c}'), j', 1}$  where  $h(\mathbf{u}_{z_{i,1}}) = \mathbf{c}\mathbf{c}'$ . By (I2),  $\mathbf{u}_{z_{i,1}}$  is identical to the first variable block in  $\mathbf{u}_{z_{i,2}}$  and thus  $h(\mathbf{u}_{z_{i,1}}) = \mathbf{c}$ . By definition if  $I^g, J^g$  contains  $\text{eq}(b_{R(\mathbf{c}), j}, b_{R(\mathbf{c}), j, R'(\mathbf{c}'), j', 1})$  and we are done.

It can now be verified that the application of the latter rule adds to  $J^g$  the fact that corresponds to  $P^{\mu_0}(h(\mathbf{x}'_0))$ .  $\square$

DLog-rewritability of CSPs is NP-complete [6, 21] and it was observed in [13] that this result lifts to generalized CSPs. It thus follows from Theorems 1 and 2 and Lemma 6 that DLog-rewritability of Boolean MDDLog programs that have equality is decidable in 2NEXPTIME. It is straightforward to verify that the 2NEXPTIME lower bound for Datalog-rewritability of MDDLog programs from [17] applies also to programs that have equality.

**Theorem 5.** *For Boolean MDDLog programs that have equality, Datalog-rewritability is 2NEXPTIME-complete.*

MDDLog programs obtained from OMQs typically do not have equality. By Lemma 5, though, we also obtain a sound but possibly incomplete algorithm for deciding DLog-rewritability of unrestricted MDDLog programs. We conjecture that this algorithm is actually complete. Note that for CSPs, it is known that adding equality preserves Datalog-rewritability [33]. Completeness of our algorithm is equivalent to an analogous result holding for MDDLog.

## 6.2 Canonical Datalog-Rewritings

For constructing actual DLog-rewritings instead of only deciding their existence, *canonical Datalog programs* play an important role. Feder and Vardi show that

for every CSP template  $T$  and all  $\ell, k > 0$ , it is straightforward to construct an  $(\ell, k)$ -Datalog program that is canonical for  $T$  in the sense that if there is any  $(\ell, k)$ -Datalog program which is equivalent to the complement of  $T$ , then the canonical one is [26]. In this section, we show that there are similarly simple canonical Datalog programs for MDDLog. Note that the existence of canonical Datalog programs for MMSNP (and thus for Boolean MDDLog) is already known from [15]. However, the construction given there is more general and rather complex, proceeding via an infinite template and exploiting that it is  $\omega$ -categorical. This makes it hard to analyze the exact structure and size of the resulting canonical programs. Here, we define canonical Datalog programs for Boolean MDDLog programs in a more elementary way.

Let  $0 \leq \ell < k$ , and let  $\Pi$  be a Boolean MDDLog program over EDB schema  $\mathbf{S}_E$  and with IDB relations from  $\mathbf{S}_I$ . We first convert  $\Pi$  into a DDLLog program  $\Pi'$  that is equivalent to  $\Pi$  on instances of treewidth  $(\ell, k)$ . Unlike  $\Pi$ , the new program  $\Pi'$  is no longer monadic. The construction of  $\Pi'$  is similar in spirit to the first step of converting an MDDLog program into simple form, c.f. Appendix A.1. To describe it, we need a preliminary.

With every DDLLog rule  $p(\mathbf{y}) \leftarrow q(\mathbf{x})$  where  $q(\mathbf{x})$  is of treewidth  $(\ell, k)$  and every  $(\ell, k)$ -tree decomposition  $(T, (B_v)_{v \in V})$  of  $q(\mathbf{x})$ , we associate a set of *rewritten rules* constructed as follows. Choose a root  $v_0$  of  $T$ , thus inducing a direction on the undirected tree  $T$ . We write  $v \prec v'$  if  $v'$  is a successor of  $v$  in  $T$  and use  $\mathbf{x}_{v'}$  to denote  $B_v \cap B_{v'}$ . For all  $v \in V \setminus \{v_0\}$  such that  $|\mathbf{x}_v| = m$ , introduce a fresh  $m$ -ary IDB relation  $Q_v$ ; note that  $m \leq \ell$ . Now, the set of rewritten rules contains one rule for each  $v \in V$ . For  $v \neq v_0$ , the rule is

$$p_v(\mathbf{y}_v) \vee Q_v(\mathbf{x}_v) \leftarrow q(\mathbf{x})|_{B_v} \wedge \bigwedge_{v \prec v'} Q_{v'}(\mathbf{x}_{v'})$$

where  $p_v(\mathbf{y}_v)$  is the sub-disjunction of  $p(\mathbf{y})$  that contains all disjuncts  $P(\mathbf{z})$  with  $\mathbf{z} \subseteq B_v$ . For  $v_0$ , we include the same rule, but use only  $p_v(\mathbf{y}_v)$  as the head. The set of rewritten rules associated with  $p(\mathbf{y}) \leftarrow q(\mathbf{x})$  is obtained by taking the union of the rewritten rules associated with  $p(\mathbf{y}) \leftarrow q(\mathbf{x})$  and any  $(T, (B_v)_{v \in V})$ .

The DDLLog program  $\Pi'$  is constructed from  $\Pi$  as follows:

1. first extend  $\Pi$  with all rules that can be obtained from a rule in  $\Pi$  by identifying variables;
2. then delete all rules with  $q(\mathbf{x})$  not of treewidth  $(\ell, k)$  and replace every rule  $p(\mathbf{y}) \leftarrow q(\mathbf{x})$  with  $q(\mathbf{x})$  of treewidth  $(\ell, k)$  with the rewritten rules associated with it.

It can be verified that  $\Pi'$  satisfies the following conditions:

- (I)  $\Pi'$  is sound for  $\Pi$ , that is, for all  $\mathbf{S}_E$ -instances  $I$ ,  $I \models \Pi'$  implies  $I \models \Pi$ ;
- (II)  $\Pi'$  is complete for  $\Pi$  on  $\mathbf{S}_E$ -instances of treewidth  $(\ell, k)$ , that is, for all such instances  $I$ ,  $I \models \Pi$  implies  $I \models \Pi'$ .

Note that  $\Pi'$  is not unconditionally complete for  $\Pi$ . For example, if  $\Pi$  consists of only a goal rule whose rule body is a  $k + 1$ -clique (without reflexive loops), then  $\Pi'$  will return false on the instance that consists of the same clique.

Let  $\mathbf{S}'_I$  denote the additional IDB relations in  $\Pi'$ . We now construct the canonical  $(\ell, k)$ -DLog program  $\Gamma^c$  for  $\Pi$ . Fix constants  $a_1, \dots, a_\ell$ . For  $\ell' \leq \ell$ , we use  $\mathfrak{J}_{\ell'}$  to denote the set of all  $\mathbf{S}_I \cup \mathbf{S}'_I$ -instances with domain  $\mathbf{a}_{\ell'} := a_1, \dots, a_{\ell'}$ . The program uses  $\ell'$ -ary IDB relations  $P_M$ , for all  $\ell' \leq \ell$  and all  $M \subseteq \mathfrak{J}_{\ell'}$ . It contains all rules  $q(\mathbf{x}) \rightarrow P_M(\mathbf{y})$ ,  $M \subseteq \mathfrak{J}_{\ell'}$ , that satisfy the following conditions:

1.  $q(\mathbf{x})$  contains at most  $k$  variables;
2. for every extension  $J$  of the  $\mathbf{S}_E$ -instance  $I_q|_{\mathbf{S}_E}$  with  $\mathbf{S}_I \cup \mathbf{S}'_I$ -facts such that
  - (a)  $J$  satisfies all rules of  $\Pi'$  and does not contain  $\text{goal}()$  and
  - (b) for each  $P_N(\mathbf{z}) \in q$ ,  $N \subseteq \mathfrak{J}_{\ell''}$ , there is an  $L \in N$  such that  $L[\mathbf{z}/\mathbf{a}_{\ell''}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{z}}$
 there is an  $L \in M$  such that  $L[\mathbf{y}/\mathbf{a}_{\ell'}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{y}}$

where  $L[\mathbf{x}/\mathbf{a}]$  denotes the result of replacing the constants in  $\mathbf{a}$  with the variables in  $\mathbf{x}$  (possibly resulting in identifications) and where  $J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{x}}$  denotes the simultaneous restriction of  $J$  to schema  $\mathbf{S}_I \cup \mathbf{S}'_I$  and constants  $\mathbf{x}$ .<sup>2</sup> We also include all rules of the form  $P_\emptyset(\mathbf{x}) \rightarrow \text{goal}()$ ,  $P_\emptyset$  of any arity from 0 to  $\ell$ . The following are central properties of canonical DLog programs.

**Lemma 7.**

1.  $\Gamma^c$  is sound for  $\Pi$ ;
2.  $\Gamma^c$  is complete for  $\Pi$  on instances of treewidth  $(\ell, k)$ .

**Proof.** For Point 1, let  $I$  be an  $\mathbf{S}_E$ -instance with  $I \models \Gamma^c$ . It suffices to show that  $I \models \Pi'$ . Let  $I = I_1, I_2, \dots$  be the sequence of  $\mathbf{S}_E \cup \mathbf{S}_I \cup \mathbf{S}'_I$ -instances obtained by chasing  $I$  with  $\Gamma^c$ , that is, by applying the rules in  $\Gamma^c$  in a fair but otherwise unspecified order. We first note that the following can be proved by induction on  $i$  (and using the definition of  $\Gamma^c$ ):

**Claim.** If  $P_M(\mathbf{b}) \in I_i$ ,  $M \subseteq \mathfrak{J}_{\ell'}$ , then for every extension  $J$  of  $I$  to the relations in  $\mathbf{S}_I \cup \mathbf{S}'_I$  that satisfies all rules of  $\Pi'$  and does not contain  $\text{goal}()$ , there is an  $L \in M$  such that  $L[\mathbf{b}/\mathbf{a}_{\ell'}] = J|_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{b}}$ .

Since  $I \models \Gamma^c$ , there are  $i > 0$  and  $\mathbf{b} \subseteq \text{dom}(I)$  such that  $P_\emptyset(\mathbf{b}) \in I_i$ . By the claim, there is thus no extension  $J$  of  $I$  to the relations in  $\mathbf{S}_I \cup \mathbf{S}'_I$  that satisfies all rules of  $\Pi$  and does not contain  $\text{goal}()$ . Consequently,  $I \models \Pi'$ .

For Point 2, assume that  $I \not\models \Gamma^c$  and let  $(T, (B_v)_{v \in V})$  be an  $(\ell, k)$ -tree decomposition of  $I$ ,  $T = (V, E)$ . Then there is an extension  $J$  of  $I$  to the IDB relations in  $\Gamma^c$  such that all rules in  $\Gamma^c$  are satisfied and  $J$  contains no atom of the form  $P_\emptyset(\mathbf{b})$ .

We use  $J$  to construct an extension  $J'$  of  $I$  to the relations in  $\mathbf{S}_I \cup \mathbf{S}'_I$ . Choose a root  $v_0$  of  $T$ , thus inducing a direction on the undirected tree  $T$ . For all  $v \in V$  and successors  $v'$  of  $v$ , choose an ordering  $\mathbf{c}_{v, v'}$  of the constants in  $B_v \cap B_{v'}$  and let  $\ell_{v, v'}$  denote the number of these constants. Let  $P_{M_1}(\mathbf{c}_{v, v'}), \dots, P_{M_r}(\mathbf{c}_{v, v'})$  be all facts of this form in  $J$ . By construction of  $\Gamma^c$ , there must be at least one such

<sup>2</sup> We could additionally demand that  $M$  is minimal so that Condition 2 is satisfied, but this is not strictly required.



fact, and the fact  $P_{M_1 \cap \dots \cap M_r}(\mathbf{c}_{v,v'})$  must also be in  $J$ . Thus, we can associate with  $v, v'$  a unique minimal set  $M_{v,v'}$  so that  $P_{M_{v,v'}}(\mathbf{c}_{v,v'}) \in J$ .

The construction of  $J'$  proceeds top down over  $T$ . At all points, we will maintain the invariant that

- (\*) for all nodes  $v \in V$  and successors  $v'$  of  $v$ , there is an  $L \in M_{v,v'}$  such that  $L[\mathbf{c}_{v,v'} / \mathbf{a}_{\ell_{v,v'}}] = J' |_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{c}_{v,v'}}$ .

The construction of  $J'$  starts at the root  $v_0$  of  $T$ . There must be an extension  $J_{v_0}$  of  $I|_{B_{v_0}}$  with  $S_I \cup S'_I$ -facts such that

- (i)  $J_{v_0}$  satisfies all rules of  $\Pi$  and does not contain **goal**()
- (ii) for each  $P_M(\mathbf{b}) \in J|_{B_{v_0}}$ ,  $M \subseteq \mathfrak{I}_{\ell'}$ , there is an  $L \in M$  such that  $L[\mathbf{b} / \mathbf{a}_{\ell'}] = J_{v_0} |_{S_i \cup S'_I, \mathbf{b}}$

as, otherwise, a rule of  $\Gamma^c$  would create an atom of the form  $P_\emptyset(\mathbf{c})$  in  $J$ . Start with putting  $J' = I \cup J_{v_0}$ . Note that for each successor  $v$  of  $v_0$ , (\*) is satisfied because of Point (ii) and since  $P_{M_{v_0,v}}(a_{v_0,v}) \in J|_{B_{v_0}}$ .

We proceed top-down over  $T$ . Assume that  $v'$  is a successor of  $v$  and  $B_v$  has already been treated. There must be an extension  $J_{v'}$  of  $I|_{B_{v'}}$  with  $S_I \cup S'_I$ -facts such that

- (i)  $J_{v'}$  satisfies all rules of  $\Pi$  and does not contain **goal**(),
- (ii)  $J_{v'} |_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{c}_{v,v'}} = J' |_{\mathbf{S}_I \cup \mathbf{S}'_I, \mathbf{c}_{v,v'}}$ , and
- (iii) for each  $P_M(\mathbf{b}) \in J|_{B_{v'}}$ ,  $M \subseteq \mathfrak{I}_{\ell'}$ , there is an  $L \in M$  such that  $L[\mathbf{b} / \mathbf{a}_{\ell'}] = J_{v'} |_{S_i \cup S'_I, \mathbf{b}}$

as, otherwise, because of (\*) a rule of  $\Gamma^c$  would create an atom of the form  $P_M(\mathbf{c}_{v,v'})$  in  $J$  with  $M \subsetneq M_{v,v'}$ , in contradiction to  $M_{v,v'}$  being minimal with  $P_{M_{v,v'}}(\mathbf{c}_{v,v'}) \in J$ . Put  $J' = J' \cup J_{v'}$ . It can again be verified that (\*) is satisfied.

By construction, the instance  $J'$  does not contain **goal**() and  $(T, (B_v)_{v \in V})$  is also a tree decomposition of  $J'$ , that is, each EDB atom *and each IDB atom* of  $J'$  falls within some bag  $B_v$ . We aim to show that  $J'$  satisfies all rules of  $\Pi$ , thus  $I \not\models \Pi$  as required.

Let  $\Pi_0$  be the result of identifying variables in rules of  $\Pi$  and recall that  $\Pi'$  is obtained from  $\Pi_0$  by dropping and rewriting rules. Let  $\rho$  be a rule in  $\Pi$  and let  $h$  be a homomorphism from its body to  $J'$ . We have to show that one of the disjuncts in the head of  $\rho$  is satisfied under  $h$ .  $\Pi_0$  contains the rule  $\rho_0$  obtained from  $\rho$  by identifying all variables  $x, y$  such that  $h(x) = h(y)$ . It clearly suffices to show that one of the disjuncts in the head of  $\rho_0$  is satisfied under  $h$ . Note that  $h$  is an injective homomorphism from the body  $q(\mathbf{x})$  of  $\rho_0$  to  $J'$  which implies that  $q(\mathbf{x})$  is of treewidth  $(\ell, k)$ . Moreover, we can read off an  $(\ell, k)$ -tree decomposition  $(T', (B'_v)_{v \in V'})$  of  $q(\mathbf{x})$  from  $h$  and  $(T, (B_v)_{v \in V})$ .

In  $\Pi'$ ,  $\rho_0$  and  $(T', (B'_v)_{v \in V'})$  are rewritten into rules  $\rho_1, \dots, \rho_m$  such that no  $\rho_i$  uses a fresh IDB relation from the head of any  $\rho_j$  with  $j \geq i$  (that is, an IDB relation that does not occur in  $\Pi_0$ , of arity at most  $\ell$ ). Let  $\rho_i$  be  $q_i(\mathbf{x}_i) \rightarrow R_{i,1}(\mathbf{x}_{i,1}) \vee \dots \vee R_{i,n_i}(\mathbf{x}_{i,n_i}) \vee Q_i(\mathbf{z}_i)$  where  $R_{i,1}(\mathbf{x}_{i,1}), \dots, R_{i,n_i}(\mathbf{x}_{i,n_i})$

are disjuncts that also occur in the head of  $\rho_0$  and  $Q_i$  is a fresh IDB relation introduced by the rewriting in the case that  $i < m$  and **false** if  $i = m$  (by which we mean: there is no  $Q_i(\mathbf{z}_i)$  disjunct in the latter case). One can show by induction on  $i$  that for  $1 \leq i \leq m$ ,

1.  $R_{j,m}(h(\mathbf{x}_{j,t})) \in J'$  for some  $j \leq i$  and  $t \in \{1, \dots, n_j\}$  or
2.  $Q_i(h(\mathbf{z}_i)) \in J'$ .

To see this, assume that Point 1 is not satisfied for some  $i$ . Then Point 2 holds for all  $j < i$ . By choice of  $\rho_i$ , there is a  $v \in V$  such that  $h(\mathbf{x}_i) \subseteq B_v$ . Thus  $h$  is a homomorphism from  $q_i(\mathbf{x}_i)$  to  $J_v$ , and consequently there is a disjunct  $R(\mathbf{z})$  in the head of  $\rho_i$  such that  $R(h(\mathbf{z})) \in J_v \subseteq J'$ . This implies that one of Points 1 or 2 is satisfied for  $i$ .

Note that Point 2 cannot hold for  $i = m$  because the  $Q_m$  disjunct is not present in  $\rho_m$ . Thus there is an  $i \leq m$  such that  $R_{i,j}(h(\mathbf{x}_{i,j})) \in J'$  for some  $j$ . Since  $R_{i,j}(\mathbf{x}_{i,j})$  occurs in the head of  $\rho_0$ , we are done.  $\square$

We are now ready to show that the canonical program is indeed canonical, as detailed by the following theorem. For two Boolean DLog programs  $\Pi_1, \Pi_2$  over the same EDB schema  $\mathbf{S}_E$ , we write  $\Pi_1 \subseteq \Pi_2$  if for every  $\mathbf{S}_E$ -instance  $I$ ,  $I \models \Pi_1$  implies  $I \models \Pi_2$ .

**Theorem 6.** *Let  $\Pi$  be a Boolean MDDLog program,  $0 \leq \ell \leq k$ , and  $\Gamma^c$  the canonical  $(\ell, k)$ -DLog program for  $\Pi$ . Then*

1.  $\Gamma \subseteq \Gamma^c$  for every  $(\ell, k)$ -DLog program  $\Gamma$  that is sound for  $\Pi$ ;
2.  $\Pi$  is  $(\ell, k)$ -DLog-rewritable iff  $\Gamma^c$  is a DLog-rewriting of  $\Pi$ .

**Proof.** Let  $\mathbf{S}_E$  be the EDB schema of  $\Pi$ .

For Point 1, let  $\Gamma$  be an  $(\ell, k)$ -DLog program that is sound for  $\Pi$  and let  $I$  be an  $\mathbf{S}_E$ -instance with  $I \models \Gamma$ . From the proof tree for  $\text{goal}()$  from  $I$  and  $\Gamma$ , we can construct an  $\mathbf{S}_E$ -instance  $J$  of treewidth  $(\ell, k)$  such that  $J \models \Gamma$  and  $J \rightarrow I$ . It suffices to show that  $J \models \Gamma^c$ , which is easy: from  $J \models \Gamma$ , we obtain  $J \models \Pi$  and Point 2 of Lemma 7 yields  $J \models \Gamma^c$ .

The “if” direction of Point 2 is trivial. For the “only if” direction, assume that  $\Pi$  is  $(\ell, k)$ -DLog-rewritable and let  $\Gamma$  be a concrete rewriting. We have to show that  $\Gamma^c$  is sound and complete for  $\Pi$ . The former is Point 1 of Lemma 7. For the latter, we get  $\Pi \subseteq \Gamma$  since  $\Gamma$  is a rewriting of  $\Pi$  and  $\Gamma \subseteq \Gamma^c$  from Point 1, thus  $\Pi \subseteq \Gamma^c$  as required.  $\square$

Note that by Point 2 of Theorem 6, the canonical  $(\ell, k)$ -DLog program for an MDDLog program  $\Pi$  is interesting even if  $\Pi$  is not rewritable into an  $(\ell, k)$ -DLog program as it is the strongest sound  $(\ell, k)$ -DLog approximation of  $\Pi$ .

## 7 Non-Boolean MDDLog Programs

We lift the results about the complexity of rewritability, about canonical DLog programs, and about the shape of rewritings and obstructions from the case of

Boolean MDDLLog programs to the non-Boolean case. For all of this, a certain extension of  $(\ell, k)$ -Datalog programs with parameters plays a central role. We thus begin by introducing these extended programs.

### 7.1 Deciding Rewritability

An  $(\ell, k)$ -Datalog program with  $n$  parameters is an  $n$ -ary  $(\ell + n, k + n)$ -Datalog program in which all IDBs have arity at least  $n$  and where in every rule, all IDB atoms agree on the variables used in the last  $n$  positions (both in rule bodies and heads and including the **goal** IDB). The last  $n$  positions of IDBs are called *parameter positions*. To visually separate the parameter positions from the non-distinguished positions, we use “|” as a delimiter to replace the usual comma, writing e.g.

$$P(x_1, x_2 | y_1, y_1, y_2) \leftarrow Q(y_1 | y_1, y_1, y_2) \wedge R(x_1, y_1, y_2, x_2)$$

where  $P, Q$  are IDB,  $R$  is EDB, and there are three parameter positions. Note that, by definition, all variable positions in **goal** atoms are parameter positions.

*Example 2.* The following is an MDLog program with one parameter that returns all constants which are on an  $R$ -cycle,  $R$  a binary EDB relation:

$$\begin{aligned} P(y | x) &\leftarrow R(x | y) \\ P(z | x) &\leftarrow P(y | x) \wedge R(y, z) \\ \text{goal}(x) &\leftarrow P(x | x) \end{aligned}$$

Parameters in Datalog programs play a similar role as parameters to least fixed-point operators in FO(LFP), see for example [7] and references therein. The program in Example 2 is not definable in MDLog without parameters, which shows that adding parameters increases expressive power. Although  $(\ell, k)$ -DLog programs with  $n$  parameters are  $(\ell + n, k + n)$ -DLog programs, one should think of them as a mild generalization of  $(\ell, k)$ -programs.

A DLog program is an  $\ell$ -DLog program if it is an  $(\ell, k)$ -DLog program for some  $k$ . To lift decidability and complexity results from the Boolean to the non-Boolean case, we show that rewritability of an  $n$ -ary MDDLLog program into  $\ell$ -DLog with  $n$  parameters can be reduced to rewritability of a Boolean MDDLLog program into  $\ell$ -DLog (without parameters). Note that the case  $\ell = 0$  is about UCQ-rewritability because 0-DLog programs (with and without parameters) are an alternative presentation of UCQs. The reduction proceeds in two steps, described by subsequent Lemmas 8 and 9.

*Example 3.* The following MDDLLog program is rewritable into the MDLog program with parameters from Example 2, but not into an MDLog program without parameters:

$$\begin{aligned} P_0(x) \vee P_1(y) &\leftarrow R(x, y) \\ \text{goal}(x) &\leftarrow P_0(x) \\ P_1(y) &\leftarrow P_1(x) \wedge R(x, y) \\ \text{goal}(x) &\leftarrow P_1(x) \end{aligned}$$

The following lemma shows that, by introducing constants, we can reduce the rewritability of MDDLog programs with non-zero arity to the rewritability of Boolean MDDLog programs. Note that the presence of constants in an  $(\ell, k)$ -DLog program is not reflected in the values of  $\ell$  and  $k$ . The *rule size* of an MDDLog program is the maximum number of variable *occurrences* in any rule in the program.

**Lemma 8.** *Given an  $n$ -ary MDDLog program  $\Pi$ , one can construct Boolean MDDLog programs with constants  $\Pi_1, \dots, \Pi_m$  over the same EDB schema such that for all  $\ell, k$ ,*

1.  *$\Pi$  is rewritable into an  $(\ell, k)$ -DLog program with  $n$  parameters iff each of  $\Pi_1, \dots, \Pi_m$  is rewritable into an  $(\ell, k)$ -DLog program with constants;*
2.  *$m \leq n^n$  and the size (resp. diameter, rule size) of each program  $\Pi_i$  is bounded by the size (resp. diameter, rule size) of  $\Pi$ .*

*The construction takes time polynomial in the size of  $|\Pi_1 \cup \dots \cup \Pi_m|$ .*

**Proof.** Let  $\Pi$  be an  $n$ -ary MDDLog program over EDB schema  $\mathbf{S}_E$ . Fix a set  $C$  of  $n$  constants. For each  $\mathbf{c} \in C^n$ , we construct from  $\Pi$  a Boolean MDDLog program  $\Pi_{\mathbf{c}}$  such that for any  $\ell < k$ ,  $\Pi$  is  $(\ell, k)$ -DLog rewritable iff all programs  $\Pi_{\mathbf{c}}$  are.

Let  $\mathbf{c} \in C^n$ . Given two  $n$ -tuples of terms (constants or variables)  $\mathbf{s}$  and  $\mathbf{t}$ , we write  $\mathbf{s} \preceq \mathbf{t}$  if  $t_i = t_j$  implies  $s_i = s_j$  for  $1 \leq i < j \leq n$ . We write  $\mathbf{s} \approx \mathbf{t}$  when  $\mathbf{s} \preceq \mathbf{t} \preceq \mathbf{s}$ . The program  $\Pi_{\mathbf{c}}$  is obtained from  $\Pi$  as follows:

- replace every rule  $\text{goal}(\mathbf{x}) \leftarrow q(\mathbf{x}, \mathbf{y})$  with  $\mathbf{c} \preceq \mathbf{x}$  by  $\text{goal}() \leftarrow q(\mathbf{c}, \mathbf{y})$ ;
- drop every rule  $\text{goal}(\mathbf{x}) \leftarrow q(\mathbf{x}, \mathbf{y})$  with  $\mathbf{c} \not\preceq \mathbf{x}$ .

Note that the non-goal rules in  $\Pi_{\mathbf{c}}$  are identical to those in  $\Pi$ . By converting proof trees for  $\Pi$  into proof trees for  $\Pi_{\mathbf{c}}$  and vice versa, one can show the following.

**Claim.** For all  $\mathbf{S}_E$ -instances  $I$  and  $\mathbf{a} \in \text{dom}(I)^n$  with  $\mathbf{a} \approx \mathbf{c}$ ,  $I \models \Pi(\mathbf{a})$  iff  $I[\mathbf{c}/\mathbf{a}] \models \Pi_{\mathbf{c}}$ .

We show that  $\Pi$  is rewritable into an  $(\ell, k)$ -DLog program with  $n$  parameters iff all of the constructed programs  $\Pi_{\mathbf{c}}$  are rewritable into an  $(\ell, k)$ -DLog program with constants.

Let  $\Gamma$  be an  $(\ell, k)$ -DLog program with  $n$  parameters that is a rewriting of  $\Pi$ . For each  $\mathbf{c} \in C^n$ , let  $\Gamma_{\mathbf{c}}$  be the Boolean  $(\ell, k)$ -DLog program with constants obtained from  $\Gamma$  as follows:

- replace every rule  $P(\mathbf{x}|\mathbf{y}) \leftarrow q(\mathbf{z}|\mathbf{y})$  with  $\mathbf{c} \preceq \mathbf{y}$  (and where  $P$  might be goal) by  $P(\mathbf{x}_{\mathbf{c}}) \leftarrow q(\mathbf{z}_{\mathbf{c}})$ , where  $\mathbf{v}_{\mathbf{c}}$  is the result of replacing in  $\mathbf{v}$  each variable  $y_i$  with  $c_i$ ;
- drop every rule  $P(\mathbf{x}|\mathbf{y}) \leftarrow q(\mathbf{z}|\mathbf{y})$  with  $\mathbf{c} \not\preceq \mathbf{y}$ .

By translating proof trees, it can be shown that  $(*) I \models \Gamma(\mathbf{c})$  iff  $I \models \Gamma_{\mathbf{c}}$ . It is now easy to show that  $\Gamma_{\mathbf{c}}$  is a rewriting of  $\Pi_{\mathbf{c}}$ : for every  $\mathbf{S}_E$ -instance  $I$ ,  $I \models \Pi_{\mathbf{c}}$  iff  $I \models \Pi(\mathbf{c})$  (by the claim) iff  $I \models \Gamma(\mathbf{c})$  (since  $\Gamma$  is a rewriting of  $\Pi$ ) iff  $I \models \Gamma_{\mathbf{c}}$  (by  $(*)$ ).

Conversely, let  $\Gamma_{\mathbf{c}}$  be a Boolean  $(\ell, k)$ -DLog program with constants that is a rewriting of  $\Pi_{\mathbf{c}}$ , for all  $\mathbf{c} \in C^n$ . We construct an  $(\ell, k)$ -DLog program with  $n$  parameters  $\Gamma$  as follows. For each  $\mathbf{c} \in C^n$ , fix a tuple  $\mathbf{v}$  of fresh variables such that  $\mathbf{v} \equiv \mathbf{c}$ . Let  $\Gamma_{\mathbf{c}}^v$  be the  $(\ell, k)$ -DLog program with  $n$  parameters obtained from  $\Gamma_{\mathbf{c}}$  as follows:

- (i) replace each  $c_i$  with  $v_i$ ;
- (ii) replace each non-goal IDB atom  $P(\mathbf{x})$  with the atom  $P^c(\mathbf{x} | \mathbf{v})$  (both in rule bodies and heads),  $P^c$  a fresh IDB relation;
- (iii) replace  $\text{goal}()$  with  $\text{goal}(\mathbf{v})$ .

Then  $\Gamma$  is defined as the union of all programs  $\Gamma_{\mathbf{c}}^v$ . We first argue that for every  $\mathbf{c} \in C^n$ ,  $\mathbf{S}_E$ -instance  $I$ , and  $\mathbf{a} \in \text{dom}(I)^n$  with  $\mathbf{a} \approx \mathbf{c}$ ,

- 1.  $I \models \Gamma_{\mathbf{c}}$  implies  $I[\mathbf{a}/\mathbf{c}] \models \Gamma_{\mathbf{c}}^v(\mathbf{a})$  and
- 2.  $I \models \Gamma_{\mathbf{c}}^v(\mathbf{a})$ , with  $\mathbf{a} \preceq \mathbf{c}'$ , implies  $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$ .

Point 1 can be proved by showing that, from a proof tree of  $\text{goal}()$  from  $I$  and  $\Gamma_{\mathbf{c}}$ , one can construct a proof tree of  $\text{goal}(\mathbf{a})$  from  $I[\mathbf{a}/\mathbf{c}]$  and  $\Gamma_{\mathbf{c}}^v$ . For Point 2, assume  $I \models \Gamma_{\mathbf{c}}^v(\mathbf{a})$  with  $\mathbf{a} \preceq \mathbf{c}'$ . Then  $I[\mathbf{c}/\mathbf{a}] \models (\Gamma_{\mathbf{c}'}^v)[\mathbf{c}/\mathbf{c}']$  can again be shown by manipulating proof trees. It can be verified that, by construction,  $(\Pi_{\mathbf{c}'})[\mathbf{c}/\mathbf{c}'] \subseteq \Pi_{\mathbf{c}}$ . Consequently and since  $\Gamma_{\mathbf{c}'}$  is a rewriting of  $\Pi_{\mathbf{c}'}$ ,  $J \models (\Gamma_{\mathbf{c}'})[\mathbf{c}/\mathbf{c}']$  implies  $J \models \Gamma_{\mathbf{c}}$  for all  $J$ , that is,  $(\Gamma_{\mathbf{c}'})[\mathbf{c}/\mathbf{c}']$  is contained in  $\Gamma_{\mathbf{c}}$  in the sense of query containment. Thus in particular  $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$ , as required.

It remains to show that  $\Gamma$  is a rewriting for  $\Pi$ . First assume that  $I \models \Pi(\mathbf{a})$ . Choose some  $\mathbf{c} \in C^n$  with  $\mathbf{a} \approx \mathbf{c}$ . Then  $I[\mathbf{c}/\mathbf{a}] \models \Pi_{\mathbf{c}}$  by the claim and thus  $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$  since  $\Gamma_{\mathbf{c}}$  is a rewriting of  $\Pi_{\mathbf{c}}$ . Point 1 above yields  $I[\mathbf{c}/\mathbf{a}][\mathbf{a}/\mathbf{c}] = I \models \Gamma(\mathbf{a})$ .

Now assume that  $I \models \Gamma(\mathbf{a})$ . By construction of  $\Gamma$ , then there is a  $\mathbf{c}' \in C^n$  such that  $\mathbf{a} \preceq \mathbf{c}'$  and  $I \models \Gamma_{\mathbf{c}'}^v(\mathbf{a})$ . To see this, note in particular that the different programs  $\Gamma_{\mathbf{c}}^v$  do not share any IDBs and thus do not interact in  $\Gamma$ . Choose a  $\mathbf{c} \in C^n$  with  $\mathbf{a} \approx \mathbf{c}$ . From Point 2 above, we obtain  $I[\mathbf{c}/\mathbf{a}] \models \Gamma_{\mathbf{c}}$  which yields  $I[\mathbf{c}/\mathbf{a}] \models \Pi_{\mathbf{c}}$ . This implies  $I \models \Pi(\mathbf{a})$  by the claim.  $\square$

We next show that constants can be eliminated from Boolean programs.

**Lemma 9.** *Given a Boolean MDDLog program  $\Pi_{\mathbf{c}}$  with constants over EDB schema  $\mathbf{S}_E$ , one can construct a Boolean MDDLog program  $\Pi$  over an EDB schema  $\mathbf{S}'_E$  such that*

- 1.  $\Pi_{\mathbf{c}}$  is rewritable into  $\ell$ -DLog with constants iff  $\Pi$  is rewritable into  $\ell$ -DLog, for any  $\ell$ ;
- 2. If  $\Pi_{\mathbf{c}}$  is of size  $n$  and diameter  $k$ , then the size of  $\Pi$  is  $2^{p(k \cdot \log n)}$ ; moreover, the diameter of  $\Pi$  is bounded by the rule size of  $\Pi_{\mathbf{c}}$ .

The construction takes time polynomial in the size of  $|II|$ .

**Proof.** Let  $\Pi_c$  be a Boolean MDDLog program over EDB schema  $\mathbf{S}_E$  that contains constants  $c_1, \dots, c_n$ . The program  $\Pi$  will be over EDB schema  $\mathbf{S}'_E = \mathbf{S}_E \cup \{R_1, \dots, R_n\}$  where  $R_1, \dots, R_n$  are fresh monadic relation symbols.  $\Pi$  contains all rules that can be obtained from a rule  $\rho$  in  $\Pi$  by choosing a partial function  $\delta$  that maps each term (variable or constant) in  $\rho$  to an element of  $\{1, \dots, n\}$  such that  $\delta(c_i) = i$  for each constant  $c_i$  and then, for each term  $t$  with  $\delta(t) = i$ ,

1. replacing each occurrence of  $t$  in the body of  $\rho$  with a fresh variable  $x$  and adding  $R_i(x)$ , and
2. replacing each occurrence of  $t$  in the head of  $\rho$  with some fresh variable introduced for  $t$  in Step 1.

Additionally,  $\Pi$  contains the rule  $\text{goal}() \leftarrow R_i(x), R_j(x)$ , for  $1 \leq i < j \leq n$ .

Note that the rewriting presented above, which we call *dejoining*, can be applied not only to MDDLog programs, but also to MDLog programs. Before we proceed, we make a basic observation about dejoining and its connection to a certain quotient construction. Let  $\Pi$  be an MDDLog program or an MDLog program, with constants  $c_1, \dots, c_n$ , and let  $\Pi_d$  be the result of dejoining  $\Pi$ . Let  $I$  be an  $\mathbf{S}'_E$ -instance such that  $R_i, R_j$  are disjoint whenever  $i \neq j$  and which does not contain the constants  $c_1, \dots, c_n$ . The *quotient* of  $I$  is the  $\mathbf{S}_E$ -instance  $I'$  obtained from  $I$  by replacing every  $d \in \text{dom}(I)$  with  $R_i(d) \in I$  by the constant  $c_i$  (which also results in the identification of elements in the active domain) and removing all atoms involving one of the  $R_i$  relations. By converting proof trees of  $\text{goal}()$  from  $\Pi$  into proof trees of  $\text{goal}()$  from  $\Pi_c$  and vice versa, one can show the following.

**Claim.**  $I \models \Pi$  iff  $I' \models \Pi_d$ .

We now show that  $\Pi_c$  is writable into  $\ell$ -DLog iff  $\Pi$  is.

First let  $\Gamma_c$  be an  $\ell$ -DLog rewriting of  $\Pi_c$ . Let  $\Gamma$  be obtained from  $\Gamma_c$  by dejoining all rules and adding the rule  $\text{goal}() \leftarrow R_i(x), R_j(x)$  for  $1 \leq i < j \leq n$ . Clearly,  $\Gamma$  is an  $\ell$ -DLog program. We argue that  $\Gamma$  is a rewriting of  $\Pi$ . Let  $I$  be an  $\mathbf{S}'_E$ -instance. W.l.o.g., we can assume that  $I$  does not contain  $c_1, \dots, c_n$ . If  $R_i, R_j$  are not disjoint for some  $i \neq j$ , then  $I \models \Pi$  and  $I \models \Gamma$ . Otherwise, let  $I'$  be the quotient of  $I$ . We have  $I \models \Pi$  iff  $I' \models \Pi_c$  (by the claim) iff  $I' \models \Gamma_c$  ( $\Gamma_c$  is rewriting of  $\Pi_c$ ) iff  $I \models \Gamma$  (again by the claim).

Let  $\Gamma$  be an  $\ell$ -DLog rewriting of  $\Pi$ . Let  $\Gamma_c$  be the program constructed from  $\Gamma$  by removing all rules that contain atoms of the form  $R_i(x)$  and  $R_j(x)$  with  $i \neq j$  and replacing all variables  $x$  that occur in a rule body in atoms of the form  $R_i(x)$  with  $c_i$  and removing all  $R_i$ -atoms from such rules. Clearly,  $\Gamma_c$  is an  $\ell$ -DLog program (with constants  $c_1, \dots, c_n$ ). We argue that  $\Gamma_c$  is a rewriting of  $\Pi_c$ . Let  $I$  be an  $\mathbf{S}_E$ -instance that w.l.o.g. does not contain  $c_1, \dots, c_n$  and let  $I' = I \cup \{R_1(c_1), \dots, R_n(c_n)\}$ . Note that  $I$  is the quotient of  $I'$ . Then  $I \models \Pi_c$  iff

$I' \models \Pi$  (by the claim) iff  $I' \models \Gamma$  ( $\Gamma$  is rewriting of  $\Pi$ ) iff  $I \models \Gamma_c$  (by construction of  $\Gamma_c$ ).  $\square$

We are now ready to lift the complexity results from Theorems 3 and 5 to the non-Boolean case.

**Theorem 7.** *In MDDLog,*

1. *FO-rewritability (equivalently: UCQ-rewritability) is 2NEXPTIME-complete;*
2. *rewritability into MDLog with parameters is in 3EXPTIME (and 2NEXPTIME-hard);*
3. *DLog-rewritability is 2NEXPTIME-complete for programs that have equality.*

**Proof.** Regarding Point 1, we recall that Proposition 1 also covers non-Boolean MDDLog programs and thus it suffices to consider UCQ-rewritability. Regarding Point 2, we remark that rewritability of the original MDDLog program into MDLog with parameters is equivalent to MDLog-rewritability of the CSPs ultimately constructed from it. Regarding Point 3, it can be verified that the constructions in the proofs of Lemmas 8 and 9 preserve the property of having equality. One can trace the blowups stated in Lemmas 8 and 9 as well as in Theorems 1 and 2 to verify that the constructed CSP templates do not become significantly larger in the non-Boolean case, that is, they still satisfy the bounds stated in Point 3 of Proposition 1. Thus, we obtain the same upper bounds as in the Boolean case.  $\square$

In view of Point 2, we remark that for non-Boolean MDDLog programs  $\Pi$ , MDLog with parameters is in a sense a more natural target for rewriting than MDLog without parameters. The intuitive reason is that positions in the answer to  $\Pi$  can be thought of as constants, and constants correspond to parameters. To make this a bit more precise, consider the grounding  $\Pi'$  of  $\Pi$  obtained by replacing, in every goal rule, each variable that occurs in the head by a constant. In contrast to the standard database setup (and in contrast to the proof of Lemma 8), we mean here constants that are interpreted according to the standard FO semantics, that is, different constants can denote the same element of an instance. When looking for an MDLog-rewriting of  $\Pi'$ , it is clearly very natural to admit the constants from  $\Pi'$  also in the rewriting. Now, one can verify that any such rewriting can be translated in a straightforward way into a rewriting of  $\Pi$  into MDLog with parameters, and vice versa.

We further note that MDLog with parameters enjoys similarly nice properties as standard MDLog. For example, containment is decidable. This follows from [16, 42] where generalizations of MDLog with parameters are studied, the actual parameters being represented by constants.

We also remark that Theorem 7 remains true when we admit constants in MDDLog programs. In fact, the proof of Lemma 8 goes through also when the original MDDLog program contains constants, and both the original and the newly introduced constants can then be removed by Lemma 9.

## 7.2 Canonical Datalog-Rewritings

We now turn our attention to canonical DLog-rewritings for non-Boolean MDDLog programs. Let  $\Pi$  be an  $n$ -ary MDDLog program. We associate with  $\Pi$  a *canonical  $(\ell, k)$ -DLog program with  $n$  parameters*, for any  $\ell < k$ . The construction is a refinement of the one from the Boolean case.

We start with some preliminaries. An  *$n$ -marked instance* is an instance  $I$  endowed with  $n$  (not necessarily distinct) distinguished elements  $\mathbf{c} = c_1, \dots, c_n$ . An  *$(\ell, k)$ -tree decomposition with  $n$  parameters* of an  $n$ -marked instance  $(I, \mathbf{c})$  is an  $(\ell + m, k + m)$ -tree decomposition of  $I$ ,  $m$  the number of distinct constants in  $\mathbf{c}$ , in which every bag  $B_v$  contains all constants from  $\mathbf{c}$ . An  $n$ -marked instance *has treewidth  $(\ell, k)$  with  $n$  parameters* if it admits an  $(\ell, k)$ -tree decomposition with  $n$  parameters.

We first convert  $\Pi$  into a DDLog program  $\Pi'$  that is equivalent to  $\Pi$  on instances of bounded treewidth. The construction is identical to the Boolean case (first variable identification, then rewriting) except that, in the rewriting step,

1. we use treewidth  $(\ell + n, k + n)$  in place of treewidth  $(\ell, k)$ ; consequently, the arity of the freshly introduced IDB relations may also be up to  $\ell + n$ ;
2. for **goal** rules, all head variables must occur in the root bag of the tree decomposition (they can then be treated in the same way as a Boolean **goal** rule despite the  $n$ -ary head relation).

It can be verified that  $\Pi'$  is sound for  $\Pi$  and that it is complete for  $\Pi$  on  $n$ -marked instances of treewidth  $(\ell, k)$  with  $n$  parameters in the sense that, for all such instances  $(I, \mathbf{c})$ ,  $I \models \Pi[\mathbf{c}]$  implies  $I \models \Pi'[\mathbf{c}]$ .  $\Pi'$  is not guaranteed to be complete for answers other than  $\mathbf{c}$  because of the way we treat goal rules in Point 2 above, for example when  $\Pi$  contains a rule of the form  $\text{goal}(x, y) \leftarrow A(x) \wedge B(y)$ .

Let  $\mathbf{S}'_I$  denote the additional IDB relations in the resulting program  $\Pi'$ . We now construct the canonical  $(\ell, k)$ -DLog program with  $n$  parameters  $\Gamma^c$ . Fix constants  $a_1, \dots, a_\ell, b_1, \dots, b_n$  and let  $\mathcal{I}_{\ell'+n}$  denote the set of all  $\mathbf{S}_I \cup \mathbf{S}'_I$ -instances with domain  $\mathbf{a}_{\ell'+n} := a_1, \dots, a_{\ell'}, b_1, \dots, b_n$ . The program uses  $\ell' + n$ -ary IDB relations  $P_M$ , for all  $\ell' \leq \ell$  and all  $M \subseteq \mathcal{I}_{\ell'+n}$ . It contains all rules  $q(\mathbf{x}) \rightarrow P_M(\mathbf{y} \mid \mathbf{x}_p)$ ,  $M \subseteq \mathcal{I}_{\ell'+n}$ , that satisfy the following conditions:

1.  $q(\mathbf{x})$  contains at most  $k + n$  variables;
2. in every extension  $J$  of the  $\mathbf{S}_E$ -instance  $I_q \mid \mathbf{s}_E$  with  $\mathbf{S}_I \cup \mathbf{S}'_I$ -facts such that
  - (a)  $J$  satisfies all rules of  $\Pi'$  and does not contain  $\text{goal}(\mathbf{x}_p)$  and
  - (b) for each  $P_N(\mathbf{z} \mid \mathbf{x}_p) \in q$ ,  $N \subseteq \mathcal{I}_{\ell'+n}$ , there is an  $L \in N$  such that
 
$$L[\mathbf{z}\mathbf{x}_p / \mathbf{a}_{\ell'+n}] = J[\mathbf{s}_I \cup \mathbf{S}'_I, \mathbf{z}]$$
 there is an  $L \in M$  such that  $L[\mathbf{y}\mathbf{x}_p / \mathbf{a}_{\ell'+n}] = J[\mathbf{s}_I \cup \mathbf{S}'_I, \mathbf{y}]$

We also include all rules of the form  $P_\emptyset(\mathbf{y} \mid \mathbf{x}_p) \rightarrow \text{goal}(\mathbf{x}_p)$ . This finishes the construction of  $\Gamma^c$ . It is straightforward to verify that  $\Gamma^c$  is sound for  $\Pi$ . It is complete in the same sense as  $\Pi'$ .



**Lemma 10.**  $\Gamma^c$  is sound for  $\Pi$ . It is complete for  $\Pi$  on  $n$ -marked instances of treewidth  $(\ell, k)$  with  $n$  parameters in the sense that for any such instance  $(I, \mathbf{c})$ ,  $I \models \Pi(\mathbf{c})$  implies  $I \models \Gamma^c(\mathbf{c})$ .

The proof of Lemma 10 is similar to that of Lemma 7, details are omitted. In analogy with Theorem 6, we can then obtain the following result about canonical DLog programs.

**Theorem 8.** Let  $\Pi$  be an  $n$ -ary MDDLog program,  $0 < \ell \leq k$ , and  $\Gamma^c$  the canonical  $(\ell, k)$ -DLog program with  $n$  parameters associated with  $\Pi$ . Then

1.  $\Gamma \subseteq \Gamma^c$  for every  $(\ell, k)$ -DLog program  $\Gamma$  that is sound for  $\Pi$ ;
2.  $\Pi$  is rewritable into  $(\ell, k)$ -DLog with  $n$  parameters iff  $\Gamma^c$  is a rewriting of  $\Pi$ .

Note that, as a consequence of Theorem 8, an  $n$ -ary MDDLog program  $\Pi$  is DLog-rewritable (in the standard sense, without parameters) iff the canonical  $(\ell, k)$ -DLog program with  $n$  parameters is a rewriting, for some  $\ell, k$ . When one is interested in DLog-rewritability without caring for the exact  $\ell, k$ -parameterization, canonical programs for the non-Boolean case thus behave exactly as canonical programs in the Boolean case. In particular, the reductions presented in Lemmas 8 and 9 show that if DLog-rewritability of Boolean programs turns out to be decidable (without assuming equality), then the same is true for DLog-rewritability of non-Boolean programs.

We now analyze the shape of rewritings of non-Boolean MDDLog programs. An  $(\ell, k)$ -tree decomposition with  $n$  parameters of an  $n$ -ary CQ  $q$  is an  $(\ell + n, k + n)$ -tree decomposition of  $q$  in which every bag  $B_v$  contains all answer variables of  $q$ . The treewidth with parameters of an  $n$ -ary CQ is now defined in the expected way.

**Theorem 9.** Let  $\Pi$  be an  $n$ -ary MDDLog program of diameter  $k$ . Then

1. if  $\Pi$  is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth  $(1, k)$  with  $n$  parameters;
2. if  $\Pi$  is rewritable into MDLog with  $n$  parameters, then it has an MDLog-rewriting with  $n$  parameters of diameter  $k$ .

Note that Theorem 9 is immediate from Theorem 4 and Lemmas 8 and 9 when  $k$  denotes the rule size of  $\Pi$  instead of its diameter. To get the improved version, one needs to carefully trace the construction of rewritings, starting with rewritings for the CSPs ultimately constructed and then through the proofs of Lemmas 1, 9, and 8. In particular, the constructions in Lemmas 1 and 9 interplay in a subtle way that can be exploited to improve the bound. Details are in the appendix.

As in the Boolean case, rewritings are closely related to obstructions. We define obstruction sets for MMSNP formulas with free variables and summarize the results that we obtain for them. A set of marked obstructions  $\mathcal{O}$  for an MMSNP formula  $\theta$  with  $n$  free variables over schema  $\mathbf{S}_E$  is a set of  $n$ -marked instances over the same schema such that for any  $\mathbf{S}_E$ -instance  $I$ , we have  $I \not\models \theta[\mathbf{a}]$

iff for some  $(O, \mathbf{c}) \in \mathcal{O}$ , there is a homomorphism  $h$  from  $O$  to  $I$  with  $h(\mathbf{c}) = \mathbf{a}$ . We obtain the following corollary from Point 1 of Theorem 9 in exactly the same way in which Corollary 1 is obtained from Point 1 of Theorem 4.

**Corollary 2.** *For every MMSNP formula  $\theta$  with  $n$  free variables, the following are equivalent:*

1.  $\theta$  is FO-rewritable;
2.  $\theta$  has a finite marked obstruction set;
3.  $\theta$  has a finite set of finite marked obstructions of treewidth  $(1, k)$  with  $n$  parameters.

It is interesting to note that this result can be viewed as a generalization of the characterization of obstruction sets for CSP templates with constants in terms of ‘c-acyclicity’ in [2]; our parameters correspond to constants in that paper. We now turn to MDLog-rewritability.

**Proposition 4.** *Let  $\theta$  be an MMSNP formula of diameter  $k$  with  $n$  free variables. Then  $\neg\theta$  is rewritable into an MDLog program with  $n$  parameters iff  $\theta$  has a set of marked obstructions (equivalently: finite marked obstructions) that are of treewidth  $(1, k)$  with  $n$  parameters.*

**Proof.** The “only if” direction is a consequence of Point 2 of Theorem 9 and the fact that, for any MDLog program  $\Pi \equiv \neg\theta$  with  $n$  parameters of diameter  $k$  over EDB schema  $\mathbf{S}_E$ , a proof tree for  $\text{goal}(\mathbf{c})$  from an  $\mathbf{S}_E$ -instance  $I$  and  $\Pi$  gives rise to a finite  $n$ -marked  $\mathbf{S}_E$ -instance  $(J, \mathbf{c})$  of treewidth  $(1, k)$  with  $n$  parameters that satisfies  $J \rightarrow I$ . The “if” direction is a consequence of the fact that the canonical  $(1, k)$ -DLog program with parameters associated with  $\neg\theta$  viewed as an MDDLog program is complete on inputs of treewidth  $(1, k, n)$  with  $n$  parameters in the sense of Lemma 10.  $\square$

As an illustration, it might be interesting to reconsider Example 3. The unary MDDLog program shown there is the negation of a unary MMSNP formula that has as a set of marked obstructions the set of all  $R$ -cycles on which one element is the marked element. Each of these obstructions has treewidth  $(1, 2)$  with  $n$  parameters, but not treewidth  $(1, 2)$  in the strict sense.

## 8 Ontology-Mediated Queries

We study rewritability of ontology-mediated queries, covering several standard description logics as the ontology language. We start with introducing the relevant classes of queries.

An *ontology-mediated query (OMQ)* over a schema  $\mathbf{S}_E$  is a triple  $(\mathcal{T}, \mathbf{S}_E, q)$  where  $\mathcal{T}$  is a TBox formulated in a description logic and  $q$  is a query over the schema  $\mathbf{S}_E \cup \text{sig}(\mathcal{T})$ ,  $\text{sig}(\mathcal{T})$  the set of relation symbols used in  $\mathcal{T}$ . The TBox can introduce symbols that are not in  $\mathbf{S}_E$ , which allows it to enrich the schema of the query  $q$ . As the TBox language, we use the description logic  $\mathcal{ALC}$ , its extension  $\mathcal{ALCI}$  with inverse roles, and the further extension  $\mathcal{SHI}$  of  $\mathcal{ALCI}$

with transitive roles and role hierarchies. Since all these logics admit only unary and binary relations, we assume that these are the only allowed arities in schemas throughout the section. As the actual query language, we use UCQs and CQs. The OMQ languages that these choices give rise to are denoted with  $(\mathcal{ALC}, \text{CQ})$ ,  $(\mathcal{SHI}, \text{UCQ})$ , and so on. In OMQs  $(\mathcal{T}, \mathbf{S}_E, q)$  from  $(\mathcal{SHI}, \text{UCQ})$ , we disallow superroles of transitive roles in  $q$ ; it is known that allowing such roles in the query poses serious additional complications, which are outside the scope of this paper, see e.g. [9, 28]. The semantics of an OMQ is given in terms of *certain answers*. We refer to the appendix for further details and only give an example here.

*Example 4.* Let  $Q = (\mathcal{T}, \mathbf{S}_E, q)$  be the following OMQ, formulated in  $(\mathcal{ALC}, \text{UCQ})$ :

$$\begin{aligned}\mathcal{T} &= \{ A \sqsubseteq B \sqcup C, \quad B \sqsubseteq \forall r.C, \quad C \sqsubseteq \exists r.C \} \\ \mathbf{S}_E &= \{ A, C, r \} \\ q(x) &= r(x, y) \wedge C(y)\end{aligned}$$

On the  $\mathbf{S}_E$ -instance

$$A(e_1), r(e_1, e_3),$$

the only certain answer to  $Q$  is  $e_1$ .

An OMQ  $Q = (\mathcal{T}, \mathbf{S}_E, q)$  is *FO-rewritable* if there is an FO query  $\varphi(x)$  over schema  $\mathbf{S}_E$  (and possibly involving equality), called an *FO-rewriting* of  $Q$ , such that for all  $\mathbf{S}_E$ -instances  $I$  and  $\mathbf{a} \subseteq \text{dom}(I)$ , we have  $I \models Q(\mathbf{a})$  iff  $I \models \varphi(\mathbf{a})$ . Other notions of rewritability such as UCQ-rewritability are defined accordingly.

Note that the TBox  $\mathcal{T}$  can be inconsistent with the input instance  $I$ , that is, there could be no model of  $\mathcal{T}$  and  $I$ . It can thus be a sensible alternative to work with *consistent FO-rewritability*, considering only  $\mathbf{S}_E$ -instances  $I$  that are consistent w.r.t.  $\mathcal{T}$ . This can then be complemented with rewritability of inconsistency for  $\mathcal{T}$ , that is, rewritability of the Boolean OMQ  $(\mathcal{T}, \mathbf{S}_E, \exists x A(x))$ ,  $A(x)$  a fresh concept name, which is true on an  $\mathbf{S}_E$ -instance  $I$  iff  $I$  is inconsistent with  $\mathcal{T}$ . It is not hard to prove, though, that consistent  $Q$ -rewritability can be reduced to  $Q$ -rewritability in polynomial time for all OMQ languages considered in this paper and all  $Q \in \{\text{FO}, \text{MDLog}, \text{DLog}\}$ ; see the corresponding proof for query containment in [17]. Moreover, rewritability of consistency was studied in [13] and shown to be NEXPTIME-complete for all OMQ languages considered in this paper.

The following is our main result on OMQs.

**Theorem 10.** *In all OMQ languages between  $(\mathcal{ALC}, \text{UCQ})$  and  $(\mathcal{SHI}, \text{UCQ})$ , as well as between  $(\mathcal{ALCI}, \text{CQ})$  and  $(\mathcal{SHI}, \text{UCQ})$ ,*

1. *FO-rewritability (equivalently: UCQ-rewritability) is 2NEXPTIME-complete; in fact, there is an algorithm which, given an OMQ  $Q = (\mathcal{T}, \mathbf{S}_E, q)$ , decides in time  $2^{2^{p(n_q \cdot \log n_{\mathcal{T}})}}$  whether  $Q$  is FO-rewritable;*
2. *MDLog-rewritability is in 3EXPTIME (and 2NEXPTIME-hard); in fact, there is an algorithm which, given an OMQ  $Q = (\mathcal{T}, \mathbf{S}_E, q)$ , decides in time  $2^{2^{2^{p(n_q \cdot \log n_{\mathcal{T}})}}}$  whether  $Q$  is MDLog-rewritable*

where  $n_q$  and  $n_{\mathcal{T}}$  are the size of  $q$  and  $\mathcal{T}$  and  $p$  is a polynomial.

Note that the runtime for deciding FO-rewritability stated in Theorem 10 is double exponential only in the size of the actual query  $q$  (which tends to be very small) while it is only single exponential in the size of the TBox (which can become large) and similarly for MDLog-rewritability, only one exponential higher.

The lower bounds in Theorem 10 are from [17]. We obtain the upper bounds by translating the OMQ into an equivalent MDDLog program and then applying the constructions that we have already established. In fact, it was shown in [13] that every OMQ from the languages mentioned in Theorem 10 can be converted into an equivalent MDDLog program at the expense of a single or even double exponential blowup, depending on the OMQ language. A slightly refined construction was presented in [17]; it yields the following result.

**Theorem 11 ([17]).** *For every OMQ  $Q = (\mathcal{T}, \mathbf{S}_E, q)$  from  $(\mathcal{SHI}, \text{UCQ})$ , one can construct an equivalent MDDLog program  $\Pi$  such that*

1. *the size of  $\Pi$  is bounded by  $2^{2^{p(n_q \cdot \log n_{\mathcal{T})}}}$ ;*
2. *the IDB schema of  $\Pi$  is of size at most  $2^{p(n_q \cdot \log n_{\mathcal{T})})}$ ;*
3. *the rule size of  $\Pi$  is bounded by  $n_q$*

where  $n_q$  and  $n_{\mathcal{T}}$  are the size of  $q$  and  $\mathcal{T}$  and  $p$  is a polynomial. The construction takes time polynomial in the size of  $\Pi$ .

Let  $Q$  be an OMQ from  $(\mathcal{SHI}, \text{UCQ})$ . Instead of deciding FO- or MDLog-rewritability of  $Q$ , we can decide the same problem for the MDDLog program delivered by Theorem 11. The bounds stated in Theorem 11, Lemmas 8 and 9, and Theorems 1 and 2, though, only guarantee that we obtain a CSP template with 3-exponentially many elements, which does not yields 2NEXPTIME upper bounds. However, it is possible to combine the construction underlying Theorem 11 with those underlying Lemmas 8 and 9 and Theorem 1 to construct from  $Q$  a simple MDDLog program  $\Pi_Q$  such that

1. *the size of  $\Pi_Q$  and the cardinality of  $\mathbf{S}'_E$  are bounded by  $2^{2^{p(n_q \cdot \log n_{\mathcal{T})}}}$  and the arity of relations in  $\mathbf{S}'_E$  is bounded by  $\max\{n_q, 2\}$ ;*
2. *the IDB schema of  $\Pi_Q$  is of size  $2^{p(n_q \cdot \log n_{\mathcal{T})})}$*

where  $p$  is a polynomial. In fact, this was already observed in [17] where exactly the same constructions that are also used in Lemmas 8 and 9 and Theorem 1 are considered, see the proof of Theorem 20 therein. Constructing a CSP template from this refined simple program and applying the decision procedures for rewritability of CSP templates, we obtain the upper bounds stated in Theorem 2.

We remark that it is not possible to extend Theorem 10 to description logics with functional roles or number restrictions since, in such DLs, FO-rewritability of OMQs is undecidable [13] (the proof can be adapted to MDLog-rewritability).

The results about the shape of rewritings stated in Theorem 9 (of course) also apply to the OMQ case. Note that, in Points 1 and 2 of that theorem, we

can then replace  $k$  with  $\max\{n_q, 2\}$ . Moreover, the canonical DLog programs introduced for MDDLog in Section 7 can also be utilized for OMQs via the translation underlying Theorem 11.

Regarding Datalog-rewritability of OMQs, we obtain a potentially incomplete decision procedure by combining Theorem 11 with Lemmas 8 and 9 and the algorithm from Section 6. It is possible to define a class of OMQs  $(\mathcal{T}, \mathbf{S}_E, q)$  that *have equality* and for which this procedure is complete. Roughly,  $\mathbf{S}_E$  needs to contain a relation  $\mathbf{eq}$  and  $\mathcal{T}$  enforces that for all models  $\mathcal{I}$  of  $\mathcal{T}$  and all  $(d, e) \in \mathbf{eq}^{\mathcal{I}}$ ,  $d$  and  $e$  satisfy exactly the same subconcepts of  $\mathcal{T}$  and exactly the same tree-shaped queries that can be obtained from  $q$  by identifying variables and then taking a subquery. We refrain from working out the details.

## 9 Discussion

We have clarified the decidability status and computational complexity of FO- and MDLog-rewritability in MMSNP, MDDLog, and various OMQ languages based on expressive description logics and conjunctive queries. For Datalog-rewritability, we were only able to obtain partial results, namely a sound algorithm that is complete only on a certain class of inputs and potentially incomplete in general. This raises several natural questions: is our algorithm actually complete in general? Does an analogue of Lemma 3 (that is, rewritability on high girth implies rewritability) hold for Datalog as a target language? What is the complexity of deciding Datalog-rewritability in the afore-mentioned languages? From an OMQ perspective, it would also be important to work towards more practical approaches for computing (FO-, MDLog-, and DLog-) rewritings. Given the high computational complexities involved, such approaches might have to be incomplete to be practically feasible. However, the degree/nature of incompleteness should then be characterized, and we expect the results in this paper to be helpful in such an endeavour.

**Acknowledgement.** We thank Libor Barto, Manuel Bodirsky, and Florent Madeleine for helpful discussions. The authors were funded by ERC grant 647289.

## References

1. Serge Abiteboul, Richard Hull, and Victor Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.
2. Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23, 2011.
3. Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakhar'yashev. The DL-Lite family and relations. *JAIR*, 36:1–69, 2009.
4. Albert Atserias. On digraph coloring problems and treewidth duality. *Eur. J. Comb.*, 29(4):796–820, 2008.

5. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
6. Libor Barto. The collapse of the bounded width hierarchy. *J. Log. Comput.*, 26(3):923–943, 2016.
7. Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *Proc. of LICS*. IEEE Computer Society, 2016.
8. Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *Proc. of LICS*, pages 293–304. IEEE Computer Society, 2015.
9. Meghyn Bienvenu, Thomas Eiter, Carsten Lutz, Magdalena Ortiz, and Mantas Simkus. Query answering in the description logic  $\mathcal{S}$ . In *Proc. of DL2010*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.
10. Meghyn Bienvenu, Peter Hansen, Carsten Lutz, and Frank Wolter. First order-rewritability and containment of conjunctive queries in horn description logics. In *Proc. of IJCAI*, 2016.
11. Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-order rewritability of atomic queries in horn description logics. In *Proc. of IJCAI*, 2013.
12. Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proc. of Reasoning Web*, volume 9203 of *LNCS*, pages 218–307. Springer, 2015.
13. Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014.
14. Manuel Bodirsky, Hubie Chen, and Tomás Feder. On the complexity of MMSNP. *SIAM J. Discrete Math.*, 26(1):404–414, 2012.
15. Manuel Bodirsky and Víctor Dalmau. Datalog and constraint satisfaction with infinite templates. *J. Comput. Syst. Sci.*, 79(1):79–100, 2013.
16. Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. Reasonable highly expressive query languages. In *Proc. of IJCAI2015*, pages 2826–2832. AAAI Press, 2015.
17. Pierre Bourhis and Carsten Lutz. Containment in monadic disjunctive Datalog, MMSNP, and expressive description logics. In *Proc. of KR*, 2016.
18. Andrei A. Bulatov, Andrei A. Krokhin, and Benoit Larose. Dualities for constraint satisfaction problems. In *Complexity of Constraints - An Overview of Current Research Themes*, volume 5250 of *LNCS*, pages 93–124. Springer, 2008.
19. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodríguez-Muro, and Riccardo Rosati. Ontologies and databases: The DL-Lite approach. In *Proc. of Reasoning Web 2009*, volume 5689 of *LNCS*, pages 255–356. Springer, 2009.
20. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
21. Hubie Chen and Benoit Larose. Asking the metaquestions in constraint tractability. *CoRR*, abs/1604.00932, 2016.
22. Gregory L. Cherlin, Saharon Shelah, and Niandong Shi. Universal graphs with forbidden subgraphs and algebraic closure. *Advances in Applied Mathematics*, 22:454–491, 1999.
23. Víctor Dalmau and Benoit Larose. Maltsev + datalog  $\rightarrow$  symmetric datalog. In *Proc. of LICS*, pages 297–306. IEEE Computer Society, 2008.

24. László Egri, Benoit Larose, and Pascal Tesson. Symmetric datalog and constraint satisfaction problems in logspace. In *Proc. of LICS*, pages 193–202. IEEE Computer Society, 2007.
25. Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-SHIQ plus rules. In *Proc. of AAAI*. AAAI Press, 2012.
26. Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.
27. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
28. Georg Gottlob, Andreas Pieris, and Lidia Tendera. Querying the guarded fragment with transitivity. In *Proc. of ICALP2013*, volume 7966 of *LNCS*, pages 287–298. Springer, 2013.
29. Peter Hansen, Carsten Lutz, İnanç Seylan, and Frank Wolter. Efficient query rewriting in the description logic  $\text{el}$  and beyond. In *Proc. of IJCAI*, 2015.
30. Mark Kaminski, Yavor Nenov, and Bernardo Cuenca Grau. Computing datalog rewritings for disjunctive datalog programs and description logic ontologies. In *Proc. of RR*, pages 76–91, 2014.
31. Gábor Kun. Constraints, MMSNP and expander relational structures. *Combinatorica*, 33(3):335–347, 2013.
32. Benoit Larose, Cynthia Loten, and Claude Tardif. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007.
33. Benoit Larose and László Zádori. Bounded width problems and algebras. *Algebra universalis*, 56(3):439–466, 2007.
34. Florent R. Madelaine. Universal structures and the logic of forbidden patterns. *Logical Methods in Computer Science*, 5(2), 2009.
35. Florent R. Madelaine. On the containment of forbidden patterns problems. In *Proc. of CP2010*, volume 6308 of *LNCS*, pages 345–359. Springer, 2010.
36. Florent R. Madelaine and Iain A. Stewart. Constraint satisfaction, logic and forbidden patterns. *SIAM J. Comput.*, 37(1):132–163, 2007.
37. Jaroslav Nešetřil. Many facets of dualities. In *Proc. of Workshop on Combinatorial Optimization*, pages 285–302. Springer, 2008.
38. Jaroslav Nešetřil and Claude Tardif. Duality theorems for finite structures (characterising gaps and good characterisations). *J. Comb. Theory, Ser. B*, 80(1):80–97, 2000.
39. Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *JAL*, 8(2):186–209, 2010.
40. Riccardo Rosati. On conjunctive query answering in  $\mathcal{EL}$ . In *Proc. of DL*, pages 451–458, 2007.
41. Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008.
42. Sebastian Rudolph and Markus Krötzsch. Flag & check: data access with monadically defined queries. In *Proc. of PODS*, pages 151–162. ACM, 2013.
43. Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos B. Stamou. Optimising resolution-based rewriting algorithms for OWL ontologies. *J. Web Sem.*, 33:30–49, 2015.

## A Translating Boolean MDDLog to generalized CSP

### A.1 From MDDLog to Simple MDDLog

Let  $\Pi$  be a Boolean MDDLog program over schema  $\mathbf{S}_E$  and of diameter  $k$ . We first construct from  $\Pi$  an equivalent Boolean MDDLog program  $\Pi_B$  such that the following conditions are satisfied:

- (i) all rule bodies are biconnected, that is, when any single variable is removed from the body (by deleting all atoms that contain it), then the resulting rule body is still connected;
- (ii) if  $R(x, \dots, x)$  occurs in a rule body with  $R$  EDB, then the body contains no other EDB atoms.

To construct  $\Pi_B$ , we first extend  $\Pi$  with all rules that can be obtained from a rule in  $\Pi$  by identifying variables; we will refer to this step as the *collapsing step*. We then split up rules that are not biconnected into multiple rules by exhaustively executing the following rewriting steps:

- replace every rule  $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1) \wedge q_2(\mathbf{x}_2)$  where  $\mathbf{x}_1$  and  $\mathbf{x}_2$  share exactly one variable  $x$  but both contain also other variables with the rules  $p_1(\mathbf{y}_1) \vee Q(x) \leftarrow q_1(\mathbf{x}_1)$  and  $p_2(\mathbf{y}_2) \leftarrow Q(x) \wedge q_2(\mathbf{x}_2)$ , where  $Q$  is a fresh monadic IDB relation and  $p_i(\mathbf{y}_i)$  is the restriction of  $p(\mathbf{y})$  to atoms that are nullary or contain a variable from  $q_i$ ,  $i \in \{1, 2\}$ ;
- replace every rule  $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1) \wedge q_2(\mathbf{x}_2)$  where  $\mathbf{x}_1$  and  $\mathbf{x}_2$  share no variables and are both non-empty with the rules  $p_1(\mathbf{y}_1) \vee Q() \leftarrow q_1(\mathbf{x}_1)$  and  $p_2(\mathbf{y}_2) \leftarrow Q() \wedge q_2(\mathbf{x}_2)$ , where  $Q()$  is a fresh nullary IDB relation and the  $p_i(\mathbf{y}_i)$  are as above;
- replace every rule  $p(\mathbf{y}) \leftarrow R(x, \dots, x) \wedge q(\mathbf{x})$  where  $R$  is an EDB relation and  $q$  contains at least one EDB atom and the variable  $x$ , with the rules  $Q(x) \leftarrow R(x, \dots, x)$  and  $p(\mathbf{y}) \leftarrow Q(x) \wedge q(\mathbf{x})$ , where  $Q$  is a fresh monadic IDB relation.

It is easy to see that the program  $\Pi_B$  is equivalent to the original program  $\Pi$ .

We next construct from  $\Pi_B$  the desired simplification  $\Pi_S$  of  $\Pi$  by replacing, in every rule, the EDB atoms in the rule body with a single EDB atom that represents the conjunction of all atoms replaced. We thus introduce fresh EDB relations that represent conjunctions of old EDB relations. Note that there can be implications between the new EDB relations that we will have to take care of in the construction of  $\Pi_B$ .

Let  $\mathcal{Q}_\Pi$  denote the set of CQs that can be obtained from a rule body in  $\Pi_B$  by consistently renaming variables, using only variables that occur in  $\Pi_B$ . Let  $\mathbf{S}_I$  be the IDB schema of  $\Pi_B$ . For every  $q(\mathbf{x}) \in \mathcal{Q}_\Pi$ , we write  $q(\mathbf{x})|_{\mathbf{S}_E}$  to denote the restriction of  $q(\mathbf{x})$  to  $\mathbf{S}_E$ -atoms, and likewise for  $q(\mathbf{x})|_{\mathbf{S}_I}$  and IDB atoms. The EDB schema  $\mathbf{S}'_E$  of  $\Pi_S$  consists of the relations  $R_{q(\mathbf{x})|_{\mathbf{S}_E}}$ ,  $q(\mathbf{x}) \in \mathcal{Q}_\Pi$ , whose arity is the number of variables in  $q(\mathbf{x})$  (which, by construction of  $\Pi_B$ , is identical to the number of variables in  $q(\mathbf{x})|_{\mathbf{S}_E}$ ). The program  $\Pi_S$  consists of the following rules:



whenever  $p(\mathbf{y}) \leftarrow q_1(\mathbf{x}_1)$  is a rule in  $\Pi_B$ ,  $q_2(\mathbf{x}_2) \in \mathcal{Q}_\Pi$ , and  $q_1(\mathbf{x}_1) \subseteq q_2(\mathbf{x}_2)$ , then  $\Pi_S$  contains the rule  $p(\mathbf{y}) \leftarrow R_{q_2(\mathbf{x}_2)}(\mathbf{x}_2) \wedge q_1(\mathbf{x}_1)|_{\mathbf{S}_I}$

The case where  $q_1(\mathbf{x}_1)$  is identical to  $q_2(\mathbf{x}_2)$  corresponds to adapting rules in  $\Pi_B$  to the new EDB signature and the other cases take care of implications between EDB relations.

*Example 5.* Assume that  $\Pi_B$  contains the following rules, where  $A$  and  $r$  are EDB relations:

$$\begin{aligned} P(x_3) &\leftarrow A(x_1) \wedge r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1) \\ \text{goal}() &\leftarrow r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1) \wedge \\ &\quad P(x_1) \wedge P(x_2) \wedge P(x_3) \end{aligned}$$

A new ternary EDB relation  $R_{q_2}$  is introduced for the EDB body atoms of the lower rule, where  $q_2 = r(x_1, x_2) \wedge r(x_2, x_3) \wedge r(x_3, x_1)$ , and a new ternary EDB relation  $R_{q_1}$  is introduced for the upper rule,  $q_1 = A(x_1) \wedge q_2$ . Then the rules are replaced with

$$\begin{aligned} P(x_3) &\leftarrow R_{q_1}(x_1, x_2, x_3) \\ \text{goal}() &\leftarrow R_{q_2}(x_1, x_2, x_3) \wedge P(x_1) \wedge P(x_2) \wedge P(x_3) \\ \text{goal}() &\leftarrow R_{q_1}(x_1, x_2, x_3) \wedge P(x_1) \wedge P(x_2) \wedge P(x_3) \end{aligned}$$

Note that  $q_1$  implies  $q_2$ , which results in two copies of the goal rule to be generated.

Proof details for the following lemma can be found in [17]. Recall that  $k$  is the diameter of the original MDDLog program  $\Pi$ .

**Lemma 11.**

1. If  $I$  is an  $\mathbf{S}_E$ -instance and  $I'$  the corresponding  $\mathbf{S}'_E$ -instance, then  $I \models \Pi$  iff  $I' \models \Pi_S$ ;
2. If  $I'$  is an  $\mathbf{S}'_E$ -instance and  $I$  the corresponding  $\mathbf{S}_E$ -instance, then
  - (a)  $I' \models \Pi_S$  implies  $I \models \Pi$ ;
  - (b)  $I \models \Pi$  implies  $I' \models \Pi_S$  if the girth of  $I'$  exceeds  $k$ .

*Example 6.* Consider the fragments of  $\Pi = \Pi_B$  and  $\Pi_S$  from Example 5. To understand the restriction to high girth instances in Point 2b of Lemma 11, take the  $\mathbf{S}'_E$ -instance  $I'$  defined by

$$R_{q_1}(a, a', c'), R_{q_1}(b, b', a'), R_{q_1}(c, c', b').$$

The goal rules from the simplified program do not apply. But the corresponding  $\mathbf{S}_E$ -instance contains the facts

$$r(c', b'), r(c', b'), r(b', a')$$

which are not covered by any EDB atom in  $I'$ . Clearly, the goal rule of  $\Pi$  applies.

## A.2 From Simple MDDLog to Generalized CSP

Let  $\Pi$  be a simple MDDLog program over EDB schema  $\mathbf{S}_E$  and with IDB schema  $\mathbf{S}_I$ . For  $i \in \{0, 1\}$ , an  $i$ -type is a set  $t$  of relation symbols from  $\mathbf{S}_I$  of arity at most  $i$  that does not contain  $\text{goal}()$  and that satisfies all rules in  $\Pi$  which use only IDB relations of arity at most  $i$  and do not involve any EDB relations.

We build a template  $T_\theta$  for each 0-type  $\theta$ . The elements of  $T_\theta$  are exactly the 1-types that agree with  $\theta$  on nullary IDB relations.  $T_\theta$  consists of the following facts:

1.  $P()$  for each nullary  $P \in \theta$ .
2.  $P(t)$  for each 1-type  $t$  and each monadic  $P \in t$ ;
3.  $R(t_1, \dots, t_n)$  for each relation  $R \in \mathbf{S}_E$  and all 1-types  $t_1, \dots, t_n$  such that  $\Pi$  does not contain a rule

$$P(x_i) \leftarrow R(x_1, \dots, x_n) \wedge P_1(x_{i_1}) \wedge \dots \wedge P_n(x_{i_n})$$

such that  $P_j \in t_{i_j}$  for  $1 \leq j \leq n$ , and  $P \notin t_i$ .

Let  $S_\Pi = \{T_\theta \mid \theta \text{ a 0-type}\}$ . Proof details for the following lemma can be found in [26].

**Lemma 12.** *For any  $\mathbf{S}_E$ -instance  $I$ , we have  $I \models \Pi$  iff  $I \not\models S_\Pi$ .*

## B MDLog-Rewritability of Generalized CSP

**Theorem 12.** *Given a finite set of templates  $S$ , it can be decided in EXPTIME whether  $\text{coCSP}(S)$  is MDLog-rewritable.*

**Proof.** We first observe that it is possible to convert a generalised coCSP into an equivalent generalised coCSP whose templates are mutually homomorphically incomparable. This is done by first replacing the original templates by their cores and then getting rid of some of the cores in the case some of them map to others. This can easily be done in EXPTIME.

Since MDLog-rewritability of coCSPs is in EXPTIME [21] and conjunctions of MDLog-queries are known to be MDLog-expressible, it now suffices to show that if a generalized coCSP with homomorphically incomparable templates is MDLog-rewritable, then each of the related templates is as well.

Assume that the generalized coCSP for a set  $S := \{I_1, \dots, I_k\}$  of homomorphically incomparable templates is MDLog-rewritable, and let  $\Gamma$  denote the MDLog-rewriting of  $\text{coCSP}(S)$ . Consider a template  $I_j$ . We will show that  $I_j$  has a possibly infinite obstruction set consisting of finite instances of treewidth  $(1, k)$  for some fixed  $k$ . It then follows from Theorem 23 of [26] that the coCSP for  $I_j$  is MDLog-rewritable.

Let  $k$  be the maximum number of variables that occur in a single rule of  $\Gamma$ . Therefore, by the following standard argument from [26], we see that  $\text{coCSP}(S)$  has a possibly infinite obstruction set  $\mathcal{O}$  of instances of treewidth  $(1, k)$ . Indeed,

let  $J \in \text{coCSP}(S)$ , i.e.,  $J \not\rightarrow I$  holds for all templates  $I$  of  $S$ . Thus  $J \models \Gamma$ , and therefore, by the semantics of datalog, there exists a finite derivation<sup>3</sup> that corresponds to a finite instance  $K_J$  of treewidth  $(1, k)$  such that  $K_J \rightarrow J$  and  $K_J \models \Gamma$ . Let  $\mathcal{O} = \{ K_J \mid J \in \text{coCSP}(S) \}$ .

Let  $\mathcal{O}(I_j)$  denote the set of exactly all finite instances of treewidth  $(1, k)$  that do not homomorphically map to  $I_j$ . We will show that  $\mathcal{O}(I_j)$  is an obstruction set for  $I_j$ . As argued above, this suffices for concluding the proof.

Assume this is not the case. Therefore there exists an instance  $D$  such that one of the following conditions hold.

- $A \rightarrow D \rightarrow I_j$  for some instance  $A$  in  $\mathcal{O}(I_j)$ .
- $A \not\rightarrow D \not\rightarrow I_j$  for each instance  $A$  in  $\mathcal{O}(I_j)$ .

We can immediately rule out the first condition above, because by definition the structures in  $\mathcal{O}(I_j)$  should not map homomorphically to  $I_j$ . Thus there exists an instance  $D \not\rightarrow I_j$  such that  $A \not\rightarrow D$  for each  $A \in \mathcal{O}(I_j)$ . We will next argue that therefore the following condition—to be called *condition*  $(\star)$  below—holds.

$(\star)$  For all instances  $B$  of treewidth  $(1, k)$ , if  $B \rightarrow D$ , then  $B \rightarrow I_j$ .

Assume for contradiction that there exists an instance  $B$  of treewidth  $(1, k)$  such that  $B \rightarrow D$  and  $B \not\rightarrow I_j$ . Thus  $B \in \mathcal{O}(I_j)$ . Therefore, since  $A \not\rightarrow D$  for each  $A \in \mathcal{O}(I_j)$ , we have  $B \not\rightarrow D$ , which is a contradiction. Thus  $(\star)$  holds.

Now consider the disjoint union  $(D \uplus I_j)$ . We have  $(D \uplus I_j) \not\rightarrow I_j$ , and furthermore, we have  $I_j \not\rightarrow I_i$  for all templates  $I_i \neq I_j$  of  $S$ . Therefore  $(D \uplus I_j)$  does not map homomorphically to any of the templates of  $S$ , and thus there exists an instance  $C \rightarrow (D \uplus I_j)$  of treewidth  $(1, k)$  in the obstruction set  $\mathcal{O}$  of  $S$ . Since  $C$  has treewidth  $(1, k)$ , the condition  $(\star)$  tells us that the connected components of  $C$  that map to  $D$  must also map to  $I_j$ . Therefore  $C \rightarrow I_j$ . However, we have  $C \not\rightarrow I_i$  for each  $I_i$  in  $S$  (including  $I_j$ ) simply because  $C$  belongs to the obstruction set  $\mathcal{O}$  of  $S$ . Thus  $C \not\rightarrow I_j$  and  $C \rightarrow I_j$ , a contradiction.  $\square$

## C Proof of Theorem 9

We restate the theorem for convenience.

**Theorem 9.** Let  $\Pi$  be an  $n$ -ary MDDLog program of diameter  $k$ . Then

1. if  $\Pi$  is FO-rewritable, then it has a UCQ-rewriting in which each CQ has treewidth  $(1, k)$  with  $n$  parameters;
2. if  $\Pi$  is rewritable into MDLog with parameters, then it has an MDLog-rewriting with  $n$  parameters of diameter  $k$ .

---

<sup>3</sup> elsewhere we speak of proof trees!

**Proof.** We treat the two cases, FO-rewritability and MDLog-rewritability with parameters, in parallel in a uniform way. To achieve uniformity, recall that FO-rewritability coincides with UCQ-rewritability by Proposition 1 and observe that a UCQ-rewriting of relaxed treewidth  $(1, k)$  can be converted into a non-recursive MDLog-rewriting with  $n$  parameters of diameter  $k$  and vice versa. We work with the latter.

Assume that an  $n$ -ary MDDLLog program  $\Pi$  over EDB schema  $\mathbf{S}_E$  is rewritable into (non-recursive) MDLog with  $n$  parameters. We can convert

1.  $\Pi$  into a Boolean MDDLLog programs  $\Pi_1, \dots, \Pi_k$  with constants (Lemma 8),
2.  $\Pi_1, \dots, \Pi_k$  into Boolean MDDLLog programs  $\Pi'_1, \dots, \Pi'_k$  without constants (Lemma 9),
3.  $\Pi'_1, \dots, \Pi'_k$  into simple Boolean MDDLLog programs  $\Pi''_1, \dots, \Pi''_k$  (Theorem 1 and Lemma 1), and
4.  $\Pi''_1, \dots, \Pi''_k$  into CSP templates  $T_1, \dots, T_k$  (Theorem 2)

such that all these programs and (complements of) templates are rewritable into (non-recursive) MDLog. Moreover, in the proofs of the mentioned lemmas and theorems, it is shown how to construct (non-recursive) MDLog-rewritings of  $\Pi''_1, \dots, \Pi''_k$  from given ones of  $T_1, \dots, T_k$ , for  $\Pi'_1, \dots, \Pi'_k$  from given ones of  $\Pi''_1, \dots, \Pi''_k$ , and so on. We are going to analyze these constructions in more detail.

We first note that for any (non-recursive) MDLog-rewritable CSP, there is a (non-recursive) MDLog-rewriting where every rule body has at most one EDB atom that contains all variables which occur in the rule body. Since each program  $\Pi''_i$  is actually *equivalent* to the complement of the CSP template  $T_i$  in Step 4, the same is true for the programs  $\Pi''_i$ . Thus, there is a (non-recursive) MDLog-rewriting  $\Gamma''_i$  of  $\Pi''_i$  in which

- (†) each rule body has at most one EDB atom that contains all variables.

The translation of  $\Pi'_i$  into  $\Pi''_i$  in Step 3 involves replacing the EDB schema  $\mathbf{S}_E$  with an aggregation schema  $\mathbf{S}'_E$ . More precisely,  $\mathbf{S}'_E$  consists of relations  $R_{q(x)}$  where  $q(x)$  is obtained from a rule body in  $\Pi'_i$  by first identifying variables, then splitting up the body into biconnected components, and finally dropping all IDB relations. When translating the rewriting  $\Gamma''_i$  of  $\Pi''_i$  into a rewriting  $\Gamma'_i$  of  $\Pi'_i$ , this change in schema is reverted. By (†), the diameter of  $\Gamma'_i$  is thus bounded by the arity of relations in  $\Gamma''_i$  and that arity, in turn, is bounded by the diameter of  $\Pi'_i$ . What's more important, though, is that we actually know what the rule bodies in  $\Gamma'_i$  look like:

- (‡) every rule body in  $\Gamma'_i$  is obtained from a rule body in  $\Pi'_i$  by first identifying variables, then splitting up the body into biconnected components, then dropping all IDB relations, and finally decorating with some fresh IDB relations without introducing fresh variables.

Now consider the translation of  $\Pi_i$  into  $\Pi'_i$  in Step 2 and the corresponding translation of  $\Gamma'_i$  into a rewriting  $\Gamma_i$  of  $\Pi_i$ . In the former, we dejoin rule bodies by (sometimes) replacing different occurrences of the same variable  $x$  with

different variables  $x_1, x_2$  and adding the atoms  $R_j(x_1)$  and  $R_j(x_2)$  for some  $j$ , thus increasing the diameter. In the latter, we rejoin the dejoined rules in  $\Pi'_i$  in the sense that we replace variables  $x, y$  with the same constant  $c_j$  whenever the rule body contains the (EDB) atoms  $R_j(x)$  and  $R_j(y)$ . It can be verified that rejoining any rule body of the form  $(\dagger)$  results in a rule body whose diameter is bounded by the diameter of  $\Pi'_i$ . This gives the desired result since Step 1 preserves diameter.  $\square$

## D Ontology-Mediated Queries and Description Logics

A  $\mathcal{ALCT}$ -concept is formed according to the syntax rule

$$C, D ::= \top \mid \perp \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists r.C \mid \exists r^-.C \mid \forall r.C \mid \forall r^-.C$$

where  $A$  ranges over a fixed countably infinite set of *concept names* and  $r$  over a fixed countably infinite set of *role names*. An  $\mathcal{ALC}$ -concept is an  $\mathcal{ALCT}$ -concept in which the constructors  $\exists r^-.C$  and  $\forall r^-.C$  are not used. An  $\mathcal{ALC}$ -TBox (resp.  $\mathcal{ALCT}$ -TBox) is a finite set of concept inclusions  $C \sqsubseteq D$ ,  $C$  and  $D$   $\mathcal{ALC}$ -concepts (resp.  $\mathcal{ALCT}$ -concepts). A  $\mathcal{SHI}$ -TBox is a finite set of

- *concept inclusions*  $C \sqsubseteq D$ ,  $C$  and  $D$   $\mathcal{SHI}$ -concepts,
- *role inclusion*  $r \sqsubseteq s$ ,  $r$  and  $s$  role names, and
- *transitivity statements*  $\text{trans}(r)$ ,  $r$  a role name.

DL semantics is given in terms of interpretations. An *interpretation* takes that form  $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$  where  $\Delta^{\mathcal{I}}$  is a non-empty set called the *domain* and  $\cdot^{\mathcal{I}}$  is the *interpretation function* which maps each concept name  $A$  to a subset  $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$  and each role name  $r$  to a binary relation  $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ . The interpretation functions is extended to concepts in the standard way, for example

$$\begin{aligned} (\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \exists e \in C^{\mathcal{I}} : (d, e) \in r^{\mathcal{I}}\} \\ (\exists r^-.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \exists e \in C^{\mathcal{I}} : (e, d) \in r^{\mathcal{I}}\}. \end{aligned}$$

We refer to standard references such as [5] for full details. An interpretation is a *model* of a TBox  $\mathcal{T}$  if it *satisfies* all statements in  $\mathcal{T}$ , that is,

- $C \sqsubseteq D \in \mathcal{T}$  implies  $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ ;
- $r \sqsubseteq s \in \mathcal{T}$  implies  $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ ;
- $\text{trans}(r) \in \mathcal{T}$  implies that  $r^{\mathcal{I}}$  is transitive.

In description logic, data is typically stored in so-called ABoxes. For uniformity with MDDLog, we use instances instead, identifying unary relations with concept names, binary relations with role names, and disallowing relations of any other arity. An interpretation  $\mathcal{I}$  is a *model* of an instance  $I$  if  $A(a) \in I$  implies  $a \in A^{\mathcal{I}}$  and  $r(a, b) \in I$  implies  $(a, b) \in r^{\mathcal{I}}$ . We say that an instance  $I$  is *consistent* with a TBox  $\mathcal{T}$  if  $I$  and  $\mathcal{T}$  have a joint model. We write  $\mathcal{T} \models r \sqsubseteq s$  if every model  $\mathcal{I}$  of  $\mathcal{T}$  satisfies  $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ .

An *ontology-mediated query (OMQ)* takes the form  $Q = (\mathcal{T}, \mathbf{S}_E, q)$  with  $\mathcal{T}$  a TBox,  $\mathbf{S}_E$  a set of concept and role names, and  $q$  a UCQ. We use  $(\mathcal{L}, \mathcal{Q})$  to refer to the set of all OMQs whose TBox is formulated in the language  $\mathcal{L}$  and where the actual queries are from the language  $\mathcal{Q}$ . For example,  $(\mathcal{ALC}, \text{UCQ})$  refers to the set of all OMQs that consist of an  $\mathcal{ALC}$ -TBox and a UCQ. For OMQs  $(\mathcal{T}, q)$  from  $(\mathcal{SHI}, \text{UCQ})$ , we adopt the following additional restriction: when  $\mathcal{T}$  contains a transitivity  $\text{trans}(r)$  and  $\mathcal{T} \models r \sqsubseteq s$ , we disallow the use of  $s$  in the query  $q$ . Let  $I$  be an  $\mathbf{S}_E$ -instance and  $\mathbf{a}$  a tuple of constants from  $I$ . We write  $I \models Q(\mathbf{a})$  and call  $\mathbf{a}$  a *certain answer to  $Q$  on  $I$*  if for all models  $\mathcal{I}$  of  $I$  and  $\mathcal{T}$ , we have  $\mathcal{I} \models q(\mathbf{a})$  (defined in the usual way).